COPYRIGHT AND USE OF THIS THESIS

This thesis must be used in accordance with the provisions of the Copyright Act 1968.

Reproduction of material protected by copyright may be an infringement of copyright and copyright owners may be entitled to take legal action against persons who infringe their copyright.

Section 51 (2) of the Copyright Act permits an authorized officer of a university library or archives to provide a copy (by communication or otherwise) of an unpublished thesis kept in the library or archives, to a person who satisfies the authorized officer that he or she requires the reproduction for the purposes of research or study.

The Copyright Act grants the creator of a work a number of moral rights, specifically the right of attribution, the right against false attribution and the right of integrity.

You may infringe the author’s moral rights if you:

- fail to acknowledge the author of this thesis if you quote sections from the work
- attribute this thesis to another author
- subject this thesis to derogatory treatment which may prejudice the author’s reputation

For further information contact the University’s Director of Copyright Services

sydney.edu.au/copyright
Methods to Support
End User Design of Arrangement-Level Musical Decision Making

Aengus Martin
2014

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Faculty of Engineering and Information Technologies
Abstract

This thesis is concerned with the study of methods and models to support the design of systems that perform music autonomously, by non-programming end users (typically musicians who are not proficient in conventional computer programming). Specifically, we address the design of the central decision making component of such systems. This component typically makes musical decisions on the time scale of a few seconds and we refer to it as a musical agent. We use the term *arrangement-level musical decision making* to refer to the activity performed by a musical agent.

We develop and characterise three separate methods for supporting the design of musical agents by non-programming end users. The first is to use the partially observable Markov decision process (POMDP) which is an elegant model of the interaction between an agent and its environment. We show its potential for designing the responses of a musical agent to particular stimuli. For example, we demonstrate how, simply by adjusting the parameters of the model, an agent’s behaviour can be varied between ‘cautious’ and ‘risk-taking’, when there is uncertainty about a musical situation. However, we identify significant challenges with regard to the use of POMDPs more generally for designing musical agents. Specifically, it is unclear what representations of musical information should be used and how to design the *reward function*, which is an important parameter of the model.

The second method is based on the paradigm of programming by example. We introduce the *similarity-based musical agent* which uses a novel instance-based machine learning algorithm to learn the style in a database of example performances. While the similarity-based musical agent shows good potential for emulating a
particular style of decision making, it requires a lot of training data: around twenty example performances were needed to emulate two relatively simple behaviours accurately. We regard this as a critical issue, envisaging that a requirement to supply so many example performances in order to create even a simple musical agent, would constitute a significant barrier to the adoption of this system.

The third method explored involves combining programming by example with a mechanism whereby a musician can embed musical knowledge into an agent. The main contribution of this work is a new piece of software called the Agent Designer Toolkit (ADTK) which supports this novel paradigm for designing musical agents. We show that the ADTK can be used to create agents that convincingly emulate styles of arrangement-level musical decision making in a wide variety of musical contexts, both mainstream and experimental, with small sets of example performances. Furthermore, the agents are often very quick to create.

The ADTK defines a novel class of models comprising a combination of variable order Markov models, association rules and other user-defined relationships between variables. To use these models in music performance, a new method was developed for computing musical decisions represented as constraint satisfaction problems subject to real-time constraints. The method is based on binary decision diagrams and it may be applicable in a variety of other real-time computer music applications.

The ADTK uniquely fulfils our goals in that it does not require the user to have any expertise in conventional computer programming. In addition, it can be seamlessly embedded in two popular music software packages so that autonomous music systems can be created entirely inside a standard music production environment. While we identify certain usability issues with the software in its current incarnation, we show the promise of a number of strategies for mitigating them. For example, we provide support for the idea that a set of widely applicable preset configurations can be identified and these will make the software much more accessible. Finally, in addition to its use in music performance, we show the potential of the ADTK for a variety of other creative uses such as the generation of new musical ideas.
Publications

Following is a list of peer-reviewed publications written over the course of this thesis-work. Those arising directly from the research herein are marked with stars (⋆):


⋆ O. Bown and A. Martin, ‘Backgammon,’ Live Performance with musical agents at the 9th ACM Conference on Creativity and Cognition, Sydney, Australia (2013)


A. Martin, C.T. Jin and O. Bown, ‘Implementation of a real-time musical
decision-maker,’ Australasian Computer Music Conference, Brisbane, Australia (2012)

K. Beilharz and A. Martin ‘The “Interface” in Site-Specific Sound Installation,’

A. Martin, C. Jin and O. Bown, ‘A Toolkit for Designing Interactive Musical
Agents,’ Proceedings of OZCHI 2011, Canberra, Australia (2011)

A. Martin and K. Beilharz, ‘Windtraces: Accessible Sonic Art,’ Proceedings of
CreateWorld, Brisbane, Australia (2011)

A. Martin, C. Jin, A. McEwan and W.L. Martens, ‘A Similarity Algorithm for
Interactive Style Imitation,’ International Computer Music Conference, Huddersfield,
UK (2011)

A. Martin, S. Ferguson, K. Beilharz, ‘Mechanisms for Controlling Complex
Sound Sources: Applications to Guitar Feedback Control,’ New Interfaces for Mu-
sical Expression, Sydney (2010)

A. Martin, C. Jin, A. van Schaik and W.L. Martens, ‘Partially Observable
Markov Decision Processes for Interactive Music Systems,’ International Computer
Acknowledgements

There are a great many people without whom the construction of this thesis would have been (i) completely impossible, (ii) much more difficult, or (iii) far less tolerable. I will leave it to the individuals themselves to figure out in which category or categories they reside.

First, I would like to thank my supervisor, Craig Jin, who from our first meeting at ICAD in Limerick, Ireland, gave me great encouragement, first to come and work as an audio programmer and then to pursue a PhD at the Computing and Audio Research Lab (CARLab) at Sydney University. Craig, along with André van Schaik and Jeffrey Shaw, was good enough to provide much of the support for said PhD from Australian Research Council Linkage Grant LP0669163, and for that I am especially grateful¹.

I would like to extend my heartfelt gratitude to Ollie Bown for becoming my associate supervisor and a hugely important influence on my work. His introduction of Ableton Live into the mix (no pun intended) was the perfect piece of the puzzle at the perfect time. My gratitude also goes to Bill Martens for his fantastic energy and understanding as Ollie’s predecessor and for seeing that Ollie would be an excellent fit.

I extend my warmest high fives and embraces (embodied, of course, in sedate handshakes) to a great many students, postdocs, research assistants, staff, interlopers and individuals of nonspecific role, from CARLab and later also from the Computing

¹I am additionally appreciative of the University of Sydney International Research Scholarship and the Norman I. Price scholarship from the School of Electrical and Information Engineering.
Engineering Laboratory at Sydney University. In particular (but in no particular
order), to Abhaya, Alan, David, Sean, Tara, Andrew, Tony, Alistair, Joubin, Roman,
Nico, Pierre, Trang, Nick, Mike, Mahendra and Gaetano, I say thanks for the many
discussions, laughs, beers and barbecues that we shared.

At a number of points during the research reported herein, I cold-called re-
searchers from fields unfamiliar to me (at the time), shamelessly looking for free
expertise. Their positive responses amazed me and helped a great deal. I would like
to thank Toby Walsh and Nina Narodytska for their invaluable advice on constraint
programming and binary decision diagrams; Kate Stevens for being an excellent
sounding board in all matters HCI; and Judy Kay for more HCI wisdom relating to
software evaluation. I would also like to thank Simon O’Callaghan for his thoughts
on a variety of machine learning issues and for having a scuba diving obsession
comparable to my own.

The contributions of practising musicians were vitally important for this work. I
would like to thank Ben Carey for offering lots of thoughtful feedback on the Agent
Designer Toolkit and for being an excellent friend and collaborator. My gratitude
also goes to the other musicians that were game to jam with nascent agent-controlled
electronics: Astrid Zeman, Evan Dorrian, James Nichols and Peter Hollo. I would
also like to thank the electronic music practitioners that participated in the studies of
the Agent Designer Toolkit, for their considerable time and effort.

I took part in a selection of interesting side projects during the course my PhD
research and there were two that provided particularly delightful diversions. I’m
grateful to Sam Ferguson for allowing me to take part in the robot feedback guitar
work and for our friendship since. I’m also hugely indebted to Kirsty Beilharz, not
only for the opportunity to co-create Windtraces, but also for being firmly behind me
at each stage of my progression to lecturer at the University of Technology, Sydney.

Finally, I would like to thank my family who took a keen interest in my progress
and enquired not a few times what exactly it was that I was doing, and Eithne, who
provided love and support every step of the way.
# Contents

Glossary  
1 Introduction  

1.1 Player-Paradigm Interactive Music Systems  
1.2 Arrangement-Level Musical Decision Making  
1.3 The Design of Player-Paradigm Systems  
  1.3.1 Computational Structure  
  1.3.2 General Design Approaches  
  1.3.3 Three Player-Paradigm Systems  
  1.3.4 Common Design Strategies  
  1.3.5 Summary  
1.4 End User Tools for Designing Musical Decision Making  
  1.4.1 Tools for Designing Structural Models  
  1.4.2 Tools for Designing Abstract Models  
  1.4.3 Generative Tools Configured by High-Level Descriptors  
  1.4.4 Conclusion  
1.5 Interactivity  
1.6 Aims of this Work  
  1.6.1 Requirements for methods for designing musical agents  
  1.6.2 Research Questions  
1.7 Outline and Contributions
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1</td>
<td>The Potential for Designing Complex Behaviour</td>
<td>81</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Designing the Reward Function</td>
<td>82</td>
</tr>
<tr>
<td>3.4</td>
<td>Conclusion</td>
<td>83</td>
</tr>
<tr>
<td>4</td>
<td>Programming Agent Behaviour by Example</td>
<td>85</td>
</tr>
<tr>
<td>4.1</td>
<td>Background</td>
<td>87</td>
</tr>
<tr>
<td>4.1.1</td>
<td>PBE for Player-Paradigm Systems</td>
<td>88</td>
</tr>
<tr>
<td>4.2</td>
<td>A PBE System for Musical Decision Making</td>
<td>90</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Overview of the Ensemble for Two Players</td>
<td>91</td>
</tr>
<tr>
<td>4.2.2</td>
<td>The Decision Making Procedure</td>
<td>94</td>
</tr>
<tr>
<td>4.2.3</td>
<td>The Similarity Algorithm</td>
<td>96</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Software Implementation</td>
<td>97</td>
</tr>
<tr>
<td>4.3</td>
<td>Evaluation Methods</td>
<td>99</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Real-time Interaction with a Musician</td>
<td>99</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Simulated Interaction</td>
<td>101</td>
</tr>
<tr>
<td>4.4</td>
<td>Results</td>
<td>102</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Observations Made During Interaction with a Musician</td>
<td>102</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Ability to Emulate the Style of Artificial Musicians</td>
<td>103</td>
</tr>
<tr>
<td>4.5</td>
<td>Discussion</td>
<td>104</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusion</td>
<td>106</td>
</tr>
<tr>
<td>5</td>
<td>The Agent Designer Toolkit</td>
<td>109</td>
</tr>
<tr>
<td>5.1</td>
<td>Outline of the Research Related to the Agent Designer Toolkit</td>
<td>110</td>
</tr>
<tr>
<td>5.2</td>
<td>Outline of this Chapter</td>
<td>115</td>
</tr>
<tr>
<td>5.3</td>
<td>Analysis of Performance Data from Three Music Systems</td>
<td>116</td>
</tr>
<tr>
<td>5.3.1</td>
<td>The Montreal System</td>
<td>117</td>
</tr>
<tr>
<td>5.3.2</td>
<td>An Electroacoustic Improvisation System</td>
<td>123</td>
</tr>
</tbody>
</table>
5.3.3 A Drum and Bass Performance System ........................................ 126
5.3.4 Analysis Outcomes ................................................................. 130
5.4 A Rationale for the Learning Components of the ADTK .................. 131
  5.4.1 Requirements of the Machine Learning Algorithms .................... 131
  5.4.2 Two Algorithms that Plausibly Meet the Requirements ................. 133
  5.4.3 Variable-Order Markov Models ............................................. 136
  5.4.4 Association Rule Learning ................................................. 142
  5.4.5 Combining Markov Models and Rules to form a Model of Music Performance 147
  5.4.6 Extending the Model with Custom Variables ............................. 149
  5.4.7 How the Model Works in an Interactive Context ........................ 162
  5.4.8 The Learning Configuration ................................................. 163
5.5 Overview of the ADTK Software .................................................. 164
5.6 Discussion .................................................................................. 166
  5.6.1 Style Emulation .................................................................... 167
  5.6.2 Discovery of New Styles ....................................................... 168
  5.6.3 Models of Music Performance ............................................... 170
  5.6.4 Interactive Machine Learning ............................................. 171
5.7 Conclusion .................................................................................. 172

6 Implementation of a Real-Time Musical Decision Maker .................... 175
  6.1 Defining the CSP for an Agent .................................................... 178
    6.1.1 A Brief Overview of Constraint Satisfaction Problems ............... 178
    6.1.2 Representing an Agent’s Musical Decision as a CSP ................. 179
  6.2 The Decision Making Procedure ................................................ 181
  6.3 Solving the dynamic CSP in real time ......................................... 185
    6.3.1 A Brief Outline of the CSP-solver implementation ................... 185
    6.3.2 Using BDDs to Solve CSPs ............................................... 186
6.3.3 Implementation of the BDD-based Fast Performer ........................................ 188
6.3.4 A Drawback of the Sugar Library ................................................................. 190
6.4 Performance of the BDD-based Fast Performer ............................................... 192
6.4.1 Results ........................................................................................................... 192
6.5 Discussion ......................................................................................................... 195
6.5.1 BDD Performance ......................................................................................... 195
6.5.2 Related Work in Computer Music ................................................................. 195
6.5.3 Alternative Approaches ................................................................................ 196
6.6 Conclusion ....................................................................................................... 197

7 Modelling Arrangement-Level Musical Decision Making with the ADTK 199
7.1 Methods ............................................................................................................ 201
7.1.1 Participants .................................................................................................. 202
7.1.2 Recording and Post-Processing Example Performances ............................... 202
7.1.3 Agents .......................................................................................................... 206
7.1.4 Agent Performances ..................................................................................... 212
7.1.5 Feedback Questionnaire ............................................................................ 213
7.2 Results: Participant Submissions and Responses ............................................. 218
7.2.1 Compositions and Examples Submitted ...................................................... 218
7.2.2 Participant Responses .................................................................................. 230
7.2.3 Direct Assessment of Agent ‘Usability’ ....................................................... 249
7.2.4 Uses of ADTK Agents ................................................................................. 251
7.3 Results: Agent Design Techniques .................................................................... 254
7.4 Discussion ......................................................................................................... 260
7.4.1 The Modelling Capabilities of ADTK Agents .............................................. 260
7.4.2 Informing the Design of a Higher-Level User Interface ......................... 264
7.4.3 Possible Sources of Error ............................................................................ 269
8 Case Studies: Agents Designed

8.1 Methodology

8.2 Case Studies

8.2.1 Automato

8.2.2 Many Piano Chops

8.2.3 Isidores

8.2.4 Cello Improvisation System

8.2.5 Backgammon

8.3 Discussion

8.3.1 The Use of the ADTK in Bricoleur-Style System Development

8.3.2 The Agent Design Workflow

8.3.3 Designing Interactive Agents

8.3.4 Reflections on the Agent Design Process

8.4 Conclusion

9 A Study of the ADTK from the Perspective of End-User Programming

9.1 Analysis of the ADTK Notation

9.1.1 The Cognitive Dimensions of Notations

9.1.2 A Cognitive Dimensions Analysis of the ADTK

9.2 A Preliminary Usability Study of the ADTK

9.2.1 Method

9.2.2 Results

9.3 Discussion

9.3.1 Attentional Investment

9.3.2 Creation of the Learning Configuration
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3.3</td>
<td>Evaluating the Agent</td>
<td>350</td>
</tr>
<tr>
<td>9.4</td>
<td>Conclusion</td>
<td>350</td>
</tr>
<tr>
<td><strong>10</strong></td>
<td>Discussion</td>
<td></td>
</tr>
<tr>
<td>10.1</td>
<td>The ADTK as a Creativity Support Tool</td>
<td>354</td>
</tr>
<tr>
<td>10.1.1</td>
<td>Design Principles</td>
<td>355</td>
</tr>
<tr>
<td>10.1.2</td>
<td>Generating New Musical Ideas</td>
<td>361</td>
</tr>
<tr>
<td>10.2</td>
<td>The ADTK in the Context of Interactive Machine Learning</td>
<td>362</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Effectively Illustrating the Learned Model</td>
<td>362</td>
</tr>
<tr>
<td>10.3</td>
<td>Summary of Future Work</td>
<td>365</td>
</tr>
<tr>
<td>10.3.1</td>
<td>Additional Components for the ADTK</td>
<td>365</td>
</tr>
<tr>
<td>10.3.2</td>
<td>HCI Research</td>
<td>368</td>
</tr>
<tr>
<td>10.4</td>
<td>Conclusion</td>
<td>369</td>
</tr>
<tr>
<td><strong>11</strong></td>
<td>Conclusion</td>
<td></td>
</tr>
<tr>
<td>11.1</td>
<td>Summary of Findings</td>
<td>371</td>
</tr>
<tr>
<td>11.1.1</td>
<td>Supplementary Findings Relating to the ADTK</td>
<td>376</td>
</tr>
<tr>
<td>11.2</td>
<td>Final Remarks</td>
<td>377</td>
</tr>
<tr>
<td><strong>Bib</strong></td>
<td>Bibliography</td>
<td></td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>Using the Agent Designer Toolkit</td>
<td></td>
</tr>
<tr>
<td>A.1</td>
<td>The Agent Designer GUI</td>
<td>401</td>
</tr>
<tr>
<td>A.1.1</td>
<td>The Top Panel</td>
<td>402</td>
</tr>
<tr>
<td>A.1.2</td>
<td>The Examples Tab</td>
<td>403</td>
</tr>
<tr>
<td>A.1.3</td>
<td>The Variables Tab</td>
<td>403</td>
</tr>
<tr>
<td>A.1.4</td>
<td>The Groups Tab</td>
<td>405</td>
</tr>
<tr>
<td>A.1.5</td>
<td>The Learn Tab</td>
<td>407</td>
</tr>
<tr>
<td>A.1.6</td>
<td>The Agent Tab</td>
<td>409</td>
</tr>
</tbody>
</table>
A.2 The ADTK in Max ......................................................... 409
  A.2.1 The Agent Designer and Fast Performer Max Externals .... 409
A.3 The ADTK in Ableton Live ............................................ 412

B Association Rule Learning ............................................ 415
Glossary


Agent Designer Device  The component of the Agent Designer Toolkit for using the Agent Designer and Fast Performer inside Ableton Live. xv, 165

Agent Designer Toolkit  The Agent Designer Toolkit. This is the set of software tools that includes the Agent Designer (for designing musical agents), the Fast Performer (for running them) and the Agent Designer Device (for using the ADTK in Ableton Live). See Section 5.5. xv, xviii, 110, 172, 200, 373

Apriori  An algorithm used to efficiently search a data set for association rules [3], i.e. an association rule learning algorithm. xvi, 284, 416, 417
**ARL** See association rule learning. 136, 142, 144, 146–148, 151, 159, 167, 169, 171, 176, 184, 185, 367, 407

**Arrangement-level musical decision making** Arrangement-level musical decision making refers to making musical decisions on the time scale of relatively short musical patterns or phrases. See Section 1.2. xvii, xix, 46, 48, 50, 63, 68–70, 83, 86, 90, 106, 110, 112, 115, 118, 130, 131, 136, 141, 142, 148, 149, 163, 164, 166, 167, 207, 261, 361, 371–374

**Association rule** An implies rule of the form $A \rightarrow B$. In this work, an association rule has three attributes in addition to the rule itself: (i) the support is the frequency with which the item set from which the rule is derived, is observed in the data, (ii) the confidence is the fraction of times the rule is correct and (iii) the size is the number of variables involved in the rule. xv, xvi, 110–112, 115, 136, 142, 146–152, 157, 161, 162, 181, 207, 208, 210, 363, 373, 374, 416, 417

**Association rule learning** The learning of association rules. Various algorithms are used to perform association rule learning, including the Apriori algorithm. xv, xvi, xx, 135, 136, 142–144, 146, 147, 150, 156, 163, 179, 189

**BDD** See binary decision diagram. 177, 178, 186–190, 192–197, 212, 313, 323, 356

**Binary decision diagram** A binary decision diagram is a representation of a Boolean function (i.e. a function of Boolean variables that evaluates to true or false). It can often be used to solve Boolean satisfiability problems very efficiently, and it has a number of other useful properties. See, e.g. [106] for a thorough overview. xvi, 177, 186, 313

**Boolean satisfiability problem** A Boolean satisfiability problem can be thought of as a constraint satisfaction problem in which all variables are Boolean (i.e. they can take only two values, true or false). Like a constraint satisfaction problem, a Boolean satisfiability problem is solved when a set of values for the variables are found that satisfies all constraints. xvi, xx, 186, 196

Constraint satisfaction problem A constraint satisfaction problem is a problem defined by a set of variables, their domains (the values they can take) and a set of constraints or relations between the variables. To solve a constraint satisfaction problem is to find a set of values for the variables that is consistent with all of the constraints. For a comprehensive overview, see, e.g. [12]. xvi, xvii, 111, 112, 170, 176, 178, 313, 322


Decision maker The component of the PQfé model that performs arrangement-level musical decision making. See Section 1.3.1. xviii, 46, 67, 70, 86, 92, 97, 99, 106, 165, 166, 176, 177, 189, 275, 276, 278, 279, 316, 318, 371

Decision point In the terminology of this thesis, decision points are the regularly spaced points in time at which an arrangement-level musical decision maker re-evaluates the situation and chooses what action to take. We also use the term to refer to the points in time at which snapshots of variable values are taken when recording a training data set of example performances. 94, 96, 103, 142, 149, 150, 156, 157, 160, 161, 163, 169, 176, 177, 179, 181, 182, 185, 186, 188, 190, 203–206, 210, 218, 220, 222, 224–226, 229, 246, 256, 263, 278, 280, 288, 295, 299, 315, 367, 403

End-user programming End-user programming is a research field relating to the extension of existing software, or the creation of new software for personal use, rather than for use by others [107]. xviii, 326, 350, 362
**EUP**  See end-user programming. 326, 327, 348

**Fast Performer**  The component of the Agent Designer Toolkit for running agents in real time. xv, 165, 166, 177, 186, 187, 189, 191, 192, 195, 197, 212, 323

**Feature extractor**  The component of the PQfe model that analyses the incoming audio or MIDI signal and extracts salient musical features from it. See Section 1.3.1. 47, 68–70, 80, 92, 162, 165, 166, 275, 277, 279, 300, 316, 318, 321, 355, 356

**General purpose**  In the context of this thesis, the term is used to refer to a method or technique that is not specific to a particular knowledge representation. A general purpose algorithm for musical decision making would embody no assumptions about the nature of the musical data involved. For example, it would (ideally) be just as applicable to choosing the parameters of digital audio effects as to choosing the musical chords to be played. Note that the term ‘general purpose’ also arises in this thesis in the context of programming languages (i.e. not specific to particular tasks or types of tasks) and CSP solvers (i.e. solvers designed to solve arbitrary CSPs). 9, 25, 31–33, 37, 38, 63, 66, 87, 90, 99, 112, 165, 371

**Generator**  The component of the PQfe model that performs ‘computer composition’ parameterised by the output of the decision maker. See Section 1.3.1. 62, 66, 69–71, 92, 117, 165, 166, 253, 275, 277, 279, 280, 282, 284–286, 316, 318, 320, 355, 356, 361, 363, 365

**Knowledge representation**  A knowledge representation is a way of representing a particular type of information inside a computer such that reasoning can be performed [156, p. 16]. xviii, 8, 9, 25, 32, 33, 37, 55, 66, 67, 69, 83, 87, 89, 90, 104, 112, 371
**Learning configuration**  The way in which the Agent Designer is configured to learn from the example performances (see Section 5.4.8, in particular, Table 5.11). This includes the priorities assigned to each variable, the VMM orders chosen, the custom variable definitions and the rule groups with their associated parameters. 163, 172, 200, 206–208, 210, 248, 249, 254, 262, 264–267, 271, 276, 282, 284, 285, 290, 295, 301, 302, 304, 306, 307, 311, 313, 318–320, 322, 356–358, 360, 361, 368, 376

**Live set**  A collection of musical material (i.e. a composition that can be performed) in Ableton Live. 47, 48, 113, 124, 127, 224, 360

**Max**  The Max interactive multimedia programming platform (see Section 2.1.2). 46, 48–50, 63, 70, 77, 97, 113, 115, 117, 164–166, 188, 201–207, 220, 254, 265, 293, 299, 354, 355, 364, 368

**MIDI**  The Musical Instrument Digital Interface (MIDI) is a protocol for transmission of control instructions between electronic musical devices. xvii, xviii, xxi, 47, 62, 69, 70, 135, 165

**Music system variable**  In the terminology of this thesis, music system variables are variables that either provide input to a musical agent or they form its output. The input music system variables represent data from the decision maker’s sensors and the output music system variables are those that the decision maker controls. xx, 142, 143, 149–151, 169, 171, 185, 191, 279, 282, 287, 291, 293–295, 299, 306

**Non-programming end user**  From the introduction to Chapter 1: We use the term *non-programming end user* to refer to users with no expertise in conventional computer programming. By *conventional* computer programming, we refer to the use of traditional computer programming languages, and not, for example, the use of markup languages, spreadsheets or macros. 4, 26, 34, 36, 46, 67, 83, 113, 114, 172, 323, 354, 358, 371

**PBE**  See programming by example. 86, 87, 90, 99, 106, 107, 116, 141, 171, 326

**Player-paradigm system**  An interactive music system intended to take the role of a human musician in real-time musical performance. The term comes from Rowe’s taxonomy of interactive music systems [152]. 67, 68, 83, 87, 88, 90, 91, 97, 358

**POMDP**  The partially observable Markov decision process (POMDP, usually pronounced *pom-dee-pee* [156, p. 626]) is a mathematical model of the interaction between an agent and its environment. 66–72, 75–78, 80–83, 372

**Programming by example**  Programming by example (PBE) is a paradigm whereby a human trains a computer or robot to perform a particular task by supplying examples of doing the task himself or herself [118]. xx, 35, 37, 38, 42, 82, 83, 86, 167, 326, 359, 372, 373

**Rule group**  In the terminology of the ADTK, rule groups are selections of variables (which can contain music system variables and custom variables) among which dependencies will be sought by association rule learning algorithms. In addition to the selection of variables, each rule group has associated with it a *minimum confidence* value, a *minimum support* value, and a *maximum itemset size* (See Section 5.4.4 and Appendix B). 143–146, 151, 159, 208, 210, 257, 259, 269, 270, 284–286, 291, 295, 298, 304, 306, 307, 313, 315, 322

**SAT**  See Boolean satisfiability problem. 186, 187, 189, 190, 196, 356
**Velocity**  An attribute of a musical note in the MIDI specification relating to the
loudness with which the note is played. 62, 89, 90

**VMM**  Variable-order Markov model. See section 2.3.2. 53, 58, 60–63, 88, 89, 110, 111,
115, 136–142, 147–149, 151, 152, 157–163, 167, 169, 170, 179, 181, 182, 184, 185,
188, 189, 191, 193, 195, 207, 208, 210, 254, 256, 258, 262, 267, 282, 291, 295, 304,
306, 311, 320, 366, 367, 373, 403, 405, 410
List of Figures

1.1 The PQfe model of a player-paradigm system ........................................ 10
1.2 Hsu’s ARHS system ............................................................................. 16
1.3 Lewis’ Voyager system ........................................................................ 17
1.4 Young’s NN Music system .................................................................... 19

2.1 The Ableton Live session view ............................................................... 48
2.2 The Ableton Live ‘Push’ controller ....................................................... 49
2.3 The standard machine learning workflow ........................................... 52
2.4 The interactive machine learning workflow ......................................... 53
2.5 A pentatonic melody ............................................................................ 55
2.6 A VMM tree ......................................................................................... 59

3.1 Schematic of the tonal improvising system ......................................... 71
3.2 The POMDP Model ............................................................................. 72
3.3 A state transition matrix of the TIS .................................................... 74
3.4 Key identification accuracy estimates ................................................ 75
3.5 Illustration of the effects of the reward function on agent behaviour .... 79

4.1 A schematic of the ensemble for two players ....................................... 93
4.2 The graphical user interface of the ensemble for two players .............. 98
4.3 Training examples for the similarity-based musical agent .................. 100
4.4 Rules underlying the systems used to test the similarity-based musical agent 102
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>Error rates of the similarity-based musical agent</td>
<td>104</td>
</tr>
<tr>
<td>5.1</td>
<td>The user interface of the Montreal system</td>
<td>118</td>
</tr>
<tr>
<td>5.2</td>
<td>Training data for the Montreal system</td>
<td>122</td>
</tr>
<tr>
<td>5.3</td>
<td>Training data for the Electroacoustic system</td>
<td>125</td>
</tr>
<tr>
<td>5.4</td>
<td>Training data for the Drum and Bass system</td>
<td>129</td>
</tr>
<tr>
<td>5.5</td>
<td>Modelling performance data with parallel Markov models.</td>
<td>137</td>
</tr>
<tr>
<td>5.6</td>
<td>How the Agent Designer components fit into the PQfe framework</td>
<td>164</td>
</tr>
<tr>
<td>5.7</td>
<td>How Ableton Live combined with the ADTK fits into the PQfe framework</td>
<td>166</td>
</tr>
<tr>
<td>5.8</td>
<td>The ‘hybrid’ machine learning workflow</td>
<td>172</td>
</tr>
<tr>
<td>6.1</td>
<td>The data representations used in the ADTK.</td>
<td>189</td>
</tr>
<tr>
<td>6.2</td>
<td>Histograms of the times taken per update for the Fast Performer</td>
<td>193</td>
</tr>
<tr>
<td>7.1</td>
<td>Examples of gestural and non-gestural parameters</td>
<td>204</td>
</tr>
<tr>
<td>7.2</td>
<td>Examples of parameter control gestures</td>
<td>205</td>
</tr>
<tr>
<td>7.3</td>
<td>The first page of the feedback questionnaire</td>
<td>214</td>
</tr>
<tr>
<td>7.4</td>
<td>A page of the feedback questionnaire for auditioning an agent</td>
<td>215</td>
</tr>
<tr>
<td>7.5</td>
<td>The final page of the feedback questionnaire</td>
<td>216</td>
</tr>
<tr>
<td>7.6</td>
<td>Responses for participant P1</td>
<td>231</td>
</tr>
<tr>
<td>7.7</td>
<td>Responses for participant P2</td>
<td>233</td>
</tr>
<tr>
<td>7.8</td>
<td>Responses for participant P3</td>
<td>237</td>
</tr>
<tr>
<td>7.9</td>
<td>Responses for participant P4</td>
<td>239</td>
</tr>
<tr>
<td>7.10</td>
<td>Responses for participant P5</td>
<td>241</td>
</tr>
<tr>
<td>7.11</td>
<td>Responses for participant P6</td>
<td>243</td>
</tr>
<tr>
<td>7.12</td>
<td>Responses for participant P7</td>
<td>244</td>
</tr>
<tr>
<td>7.13</td>
<td>The median responses across participants for each agent and each question</td>
<td>246</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.14</td>
<td>Histograms of response frequencies across participants for each agent type and each question</td>
<td>247</td>
</tr>
<tr>
<td>7.15</td>
<td>Assessments of agent usabilities</td>
<td>251</td>
</tr>
<tr>
<td>7.16</td>
<td>Correspondences between post-hoc question responses and Likert-type responses</td>
<td>252</td>
</tr>
<tr>
<td>7.17</td>
<td>Histograms of response frequencies for questions related to uses of the ADTK</td>
<td>253</td>
</tr>
<tr>
<td>7.18</td>
<td>Inappropriate interactions caused by modelling the number of active tracks</td>
<td>255</td>
</tr>
<tr>
<td>7.19</td>
<td>Modelling how tracks or instruments are layered</td>
<td>257</td>
</tr>
<tr>
<td>7.20</td>
<td>Structure imposed by rules</td>
<td>258</td>
</tr>
<tr>
<td>7.21</td>
<td>A limitation of ADTK models</td>
<td>262</td>
</tr>
<tr>
<td>7.22</td>
<td>Mock-up of Layer 2 of an Interface for the Agent Designer</td>
<td>268</td>
</tr>
<tr>
<td>8.1</td>
<td>PQfe diagram of the Automato system</td>
<td>278</td>
</tr>
<tr>
<td>8.2</td>
<td>Training data for the Automato system</td>
<td>281</td>
</tr>
<tr>
<td>8.3</td>
<td>Planned structure of the improvisation involving the Many Piano Chops system</td>
<td>287</td>
</tr>
<tr>
<td>8.4</td>
<td>Training data for the Many Piano Chops system</td>
<td>289</td>
</tr>
<tr>
<td>8.5</td>
<td>Schematic of the Isidores system</td>
<td>293</td>
</tr>
<tr>
<td>8.6</td>
<td>Training data for the Isidores system</td>
<td>296</td>
</tr>
<tr>
<td>8.7</td>
<td>Training data for the Cello Improvisation system</td>
<td>303</td>
</tr>
<tr>
<td>8.8</td>
<td>A Backgammon board ready for play</td>
<td>309</td>
</tr>
<tr>
<td>8.9</td>
<td>Training data for the Backgammon system</td>
<td>312</td>
</tr>
<tr>
<td>10.1</td>
<td>Simple statistics to improve a user’s understanding of the model</td>
<td>365</td>
</tr>
<tr>
<td>A.1</td>
<td>Screenshot of the Agent Designer after loading</td>
<td>402</td>
</tr>
<tr>
<td>A.2</td>
<td>Screenshot of the Agent Designer Examples Tab</td>
<td>404</td>
</tr>
<tr>
<td>A.3</td>
<td>Screenshot of the Agent Designer Example Edit window</td>
<td>404</td>
</tr>
<tr>
<td>A.4</td>
<td>Screenshot of the Agent Designer Variables Tab</td>
<td>406</td>
</tr>
<tr>
<td>A.5</td>
<td>Screenshot of the Agent Designer Add/Edit Custom Variable Dialog Box</td>
<td>406</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>A.6</td>
<td>Screenshot of the Agent Designer Groups Tab</td>
<td>407</td>
</tr>
<tr>
<td>A.7</td>
<td>Screenshot of the Agent Designer Add/Edit Custom Rule Group Dialog Box</td>
<td>408</td>
</tr>
<tr>
<td>A.8</td>
<td>Screenshot of the Agent Designer Learn Tab</td>
<td>408</td>
</tr>
<tr>
<td>A.9</td>
<td>Screenshot of the Agent Designer Agent Tab</td>
<td>409</td>
</tr>
<tr>
<td>A.10</td>
<td>Screenshot of a Max patch incorporating the ADTK</td>
<td>411</td>
</tr>
<tr>
<td>A.11</td>
<td>Screenshot of an Ableton Live set with the Agent Designer Device</td>
<td>412</td>
</tr>
</tbody>
</table>
## List of Tables

2.1 The state transition matrix derived from the pentatonic melody .......................... 56

3.1 The reward function parameters for two POMDP agents ................................. 78

4.1 Conditions for evaluating the similarity-based musical agent ......................... 102

5.1 The control parameters of the ensemble for two players ............................... 120

5.2 Stylistically salient patterns for the Montreal system ................................. 121

5.3 The control parameters of the *Electroacoustic Improvisation* system ........... 124

5.4 Stylistically salient patterns for the *ElectroAcoustic Improvisation* system ..... 126

5.5 The control parameters of the *Drum and Bass* system ............................... 128

5.6 Stylistically salient patterns for the *Drum and Bass* system ....................... 129

5.7 Algorithms available in commercial music software .................................. 135

5.8 The custom variable types ........................................................................... 155

5.9 Illustration of using custom variables to define fills ................................... 159

5.10 Illustration of using custom variables for hypermetrical structure .......... 160

5.11 Summary of agent design learning parameters ....................................... 163

6.1 Fast Performer update times for a variety of agents ................................... 194

7.1 The *Preset1* learning configuration ........................................................... 209

7.2 The *Preset2* learning configuration ........................................................... 211

7.3 List of feedback questions ............................................................................ 217

7.4 Summary of the Ableton Live set used by P1 ............................................ 219
List of Tables

7.5 Summary of the Max Patch used by P2 ........................................ 221
7.6 Summary of the Ableton Live set used by P3 .............................. 223
7.7 Summary of the Ableton Live set used by P4 .............................. 224
7.8 Summary of the Ableton Live set used by P5 .............................. 226
7.9 Summary of the Ableton Live set used by P6 .............................. 227
7.10 Summary of the Ableton Live set used by P7 ............................. 229
7.11 The percentage confidence scale presented to participants ............ 250

8.1 The music system variables of the Automato system .................... 279
8.2 The learning configuration for the Automato musical agent ............ 283
8.3 The music system variables of the Many Piano Chops system .......... 288
8.4 The learning configuration for the Many Piano Chops musical agent .. 292
8.5 The music system variables of the Isidores system ...................... 294
8.6 The learning configuration for the Isidores musical agent .............. 297
8.7 The music system variables of the Cello Improvisation system ......... 300
8.8 The learning configuration for the Cello Improvisation musical agent . 305
8.9 The learning configuration for the Backgammon musical agent ........ 314

9.1 The learning configuration for Agent01 ....................................... 336
9.2 The learning configuration for Agent02 ....................................... 337
9.3 The learning configuration for Agent03 ....................................... 338
9.4 The learning configuration for Agent04 ....................................... 339
9.5 The learning configuration for Agent05 ....................................... 340
9.6 The learning configuration for Agent06 ....................................... 342
9.7 The learning configuration for Agent07 ....................................... 344
9.8 The learning configuration for Agent08 ....................................... 346
Chapter 1

Introduction

For almost fifty years, musicians and researchers have been developing computer music systems that generate music in real-time, either autonomously, or in tandem with one or more human performers (see, e.g. [49, Ch. 11]). At first, such research was only possible for those with access to expensive and scarce computational resources. However, as digital computers became less costly, the field became more accessible and the introduction of the Musical Instrument Digital Interface (MIDI) standard in the early 1980’s heralded the first wave [153] of interactive music systems [152] that received and generated music, represented symbolically. It was not long before readily available computational power had increased to the extent that such systems could receive and generate digital audio signals and now a modern laptop computer can comfortably run software implementations of sophisticated, artificially intelligent musicians, along with their virtual instruments, in real-time.

Along with growth in computing power and ubiquity, the increasing sophistication of interactive and generative music systems has been supported by commensurate development of the platforms on which they were built. Early systems were implemented using low-level general purpose computer languages such as Fortran and machine-specific assembly languages. Gradually, new languages emerged that were designed for real-time musical operations, such as the Hierarchical Music Specification Language [140], and the Max graphical programming language [143, 145],
which is still widely used today. These have since been followed by a host of others such as SuperCollider [126], PureData [144], ChucK [179] and Nyquist [66].

However, despite the availability of powerful, music-oriented platforms, the design of interactive and generative music systems has largely remained the purview of musicians proficient in computer programming and algorithm development, or alternatively, musically inclined computer scientists. While there are some software tools for creating generative and interactive music systems without requiring such expertise, they are often idiosyncratic and overwhelmingly rooted in traditional concepts of western tonal music (a survey follows below).

In this thesis, we explore general purpose methods for supporting the creation of interactive and generative music systems by musicians who are non-programming end users. We use the term general purpose to refer to methods that are not specific to the concepts of western music notation. Our focus is on methods that could be used as easily to design the musical decision making behaviour of a system that controls a bank of audio effects, as to design that of a system that can play musical chords. We use the term non-programming end user to refer to users with no expertise in conventional computer programming.

More specifically, in this thesis we are concerned with the design of arrangement-level musical decision making. We introduce this term to refer to musical decision making that occurs on the time scale at which one combines relatively short musical patterns or phrases (typically a few seconds), rather than on the lower time scale of individual notes or gestures. For succinctness, we refer to a computational component that conducts arrangement-level musical decision making as a musical agent. Though this is a somewhat narrower definition of the term than that given or implied elsewhere in the computer music literature (see [57, Ch. 8] for a discussion), it is consistent with the use of the term in computer science, and it corresponds well

---

1 Researchers such as Blackwell [20, 21] have argued that it is difficult to define what constitutes programming and what does not, and that almost all computer users can be said to engage in programming to some degree. By conventional computer programming, we refer to the use of traditional computer programming languages, and not, for example, the use of markup languages, spreadsheets or macros.
to an established formalism for describing the computational structure of interactive music systems (again, see below).

Our motivation for focussing on the design of arrangement-level musical decision making, that is, the design of musical agents, is two-fold. First, real-time interactive systems typically operate on both of the time scales mentioned above (and sometimes more), and with regard to this, artists working in the field, such as Young [186] and Bown [30], have separately observed that unlike that of decision making processes on the longer time scale, the creation of decision making processes on the shorter time scale (e.g. algorithms that generate sequences of notes) is a task that has much in common with standard music composition. Thus, for musicians, there is, perhaps, a greater need for methods to support the design of decision making on the longer time scale.

The second, and more important, reason for focussing on methods for designing arrangement-level decision making is that such methods would support tools that fit directly into contemporary electronic music performance practise. It is common for contemporary electronic music practitioners to perform by sequencing and juxtaposing pre-prepared musical elements such as short audio samples or MIDI sequences. This is evidenced, for example, by the main performance interface of Ableton Live, a popular music performance and production software package. This means that once provided with a tool for designing arrangement-level musical decision making, a contemporary electronic music practitioner could apply it in the context of Ableton Live, for example, in order to readily develop an interactive or generative system without ever needing to leave the standard music production environment. This is an exciting prospect.

**How to Read this Thesis**

For an outline of the research questions addressed in this thesis, and the key findings, it should be sufficient to read this chapter and the final one (Chapter 11). We
also provide a web page of illustrative videos and sound recordings\(^2\) that should help get a broad idea of the main parts of the work. Chapters 3 and 4 report on separate preliminary studies and they can be read independently. Chapters 5-10 are concerned entirely with the development and study of the main tangible output of this research: the *Agent Designer Toolkit* software. These should be read in sequence, though Chapter 6 can be safely skipped for readers not interested in the technical details of the implementation of the software.

We also note that this thesis features frequent use of specialist terminology, both newly introduced and from a range of fields including machine learning and computer music. As an aid to the reader, there is a glossary of frequently used terms and acronyms just after the table of contents. For convenience, in the electronic version of this document, the page numbers given in the glossary (indicating where each term has been used) are clickable links. Similar ‘back links’ are provided in the bibliography.

### Outline of this Chapter

The remainder of this chapter is structured as follows. We begin with an outline of the characteristics of interactive and generative *player-paradigm* music systems (Section 1.1). We then discuss the concept of *arrangement-level musical decision making*, introduced above (Section 1.2). Following this, we survey the approaches taken in the computer music literature to the design of player-paradigm systems (Section 1.3). We continue in Section 1.4 with an overview of existing tools for designing musical decision making, aimed at non-programming end users. This is followed by a discussion of the notion of *interactivity* as it understood in the field of computer music (Section 1.5). We then give the specific aims of this work and the research questions addressed in this thesis (Section 1.6). This is followed by an outline of the remaining chapters and a succinct list of the contributions of this work (Section 1.7). We conclude with a personal statement on this thesis work (Section 1.8).

\(^2\)See: [am-process.org/thesis-examples](http://am-process.org/thesis-examples).
1.1 Player-Paradigm Interactive Music Systems

We are interested in the design of interactive and generative music systems. Rowe was among the first to provide a taxonomy of interactive computer music systems [152, pp. 6-8], much of which applies equally to generative systems. He differentiated between systems according to:

1. the extent to which they are score-driven or performance-driven (i.e. the extent to which performances are pre-planned, as opposed to improvised);

2. whether they generate output by transforming musical input, by generating new output, or by playing back pre-composed musical sequences; and finally,

3. whether they take a role closer to that of a musical instrument, or a player (i.e. the extent to which the system is controlled by the musician).

Various researchers have raised issues with this taxonomy (and others similar to it, such as that of Winkler [181, Ch. 2]) being couched in the ‘classical triumvirate of composition, performer and instrument’ [32]. For example, Bown et al [32] cite the deficiencies of this taxonomy with respect to the new interactive possibilities afforded by the software medium, such as its transferability and modifiability. However, Rowe’s taxonomy suffices to sketch the types of systems with which we are concerned in this thesis.

With respect to the first of Rowe’s axes, in this work, we focus on systems that are largely performance-driven, operating in contexts that do not require adherence to a pre-composed score. The second axis relates primarily to ways of generating musical material at the time scale of individual notes and gestures. As indicated in the chapter introduction, we are interested in the design of arrangement-level musical decision making and one aim of this work is that it be largely agnostic with respect to the specific lower-level processes that produce sequences of notes and gestures (we use the term general purpose to refer to this; see below). A musical agent operating on the time scale of arrangement-level musical decision making
might choose the parameters of transformative or generative processes operating on a shorter time scale, or it could choose individual pre-composed sequences of material, or combinations of these. Finally, with respect to the third axis, here, we focus on player-paradigm systems, that is, systems that can take the place of a musician in the context of live performance, and which exhibit some amount of autonomy, making musical decisions that may be influenced by the actions of a musician, but are not controlled by the musician (see the discussion of interactivity in Section 1.5). In short, this work is concerned with player-paradigm systems for use in improvisatory contexts, either on their own (generative) or in tandem with other musicians (interactive).

1.2 Arrangement-Level Musical Decision Making

As indicated in the introduction, we are interested specifically in methods to support the design of arrangement-level musical decision making. We defined this as decision making occurring on the time scale on which a musician combines relatively short musical patterns or phrases (i.e. on the order of a few seconds). In the following, we support the idea that this is a distinct and important type of musical decision making.

Arrangement-level musical decision making finds a parallel in Pressing’s cognitive model of musical improvisation [142, pp. 152-166]. In Pressing’s model, a musical improvisation is represented as a series of non-overlapping sections called event clusters, which are pre-defined sets of musical events. In the case of improvisation with acoustic instruments, events are usually musical notes or articulations. At frequent time points during a performance, the improviser chooses the next event cluster that will take place. The actual execution of the event cluster is deferred to lower level motor control mechanisms which operate at a speed faster than conscious decision making. Thus, arrangement-level musical decision making is similar to the conscious choosing of event clusters proposed in Pressing’s model. We also note...
that the time scale of arrangement-level musical decision making corresponds well to the granularity of the ‘subjective present’, which is related to the slower of two temporal processing mechanisms in the human brain [141].

The significance of arrangement-level musical decision making is also evidenced by the fact that it features in both experimental and commercial music software systems. First, as will be shown in the next section, many published player-paradigm systems make decisions explicitly on this time scale. Second as mentioned above, it is common in contemporary electronic music practise, to perform by combining and sequencing relatively short musical patterns, i.e., arrangement-level musical decision making. This is strongly evidenced by popular software packages such as Ableton Live, which explicitly support this activity in a live performance context. There is an overview of Ableton Live in the next chapter.

Finally, we place the time scale of arrangement-level musical decision making in the context of other time scales relevant to music. Roads [148, Ch. 1] defines a set of perceptual time scales for music:

- **The micro level** is on the order of a few milliseconds to one tenth of a second. A player-paradigm-system making decisions on this time scale is actively controlling minute timbral variations of the sound produced. In addition, this is a typical duration for a short-term Fourier transform window, and so it relevant to the analysis of incoming sound.

- **The sound object level** is on the order of ‘a fraction of a second to several seconds.’ Systems which generate musical output note by note are making decisions on this time scale.

- **The meso level** is on the order of a few seconds to a few minutes. It is the level at which sound objects are grouped into ‘hierarchies of phrase structures of various sizes’. In addition, an interactive music system that analyses its input or generates its output phrase by phrase is making decisions on this time scale.

- **The macro level** which is the time scale of the overall musical form.
Thus, the time scale on which arrangement-level musical decision making occurs corresponds to the lower end of the range given for the meso level (i.e. a few seconds or more).

1.3 The Design of Player-Paradigm Systems

A player-paradigm interactive music system makes decisions in real time about the sound it will output, with these decisions being influenced in some way by the data arriving at its input. Though we are not exclusively interested in interactive systems, it is these systems that must operate according to real-time constraints, whereas generative systems, in general, do not. Therefore, in this section we survey the approaches to designing interactive player-paradigm systems. In particular we aim to support the notions, introduced already,

- that in the design of player-paradigm systems, it is common to conceptualise musical decision making as occurring separately on at least two time scales, in particular, on the time scale of notes and gestures, and on the time scale of arrangement-level musical decision making; and
- that arrangement-level musical decision making can involve a wide variety of knowledge representations, and therefore, that a method for designing such decision making behaviour would ideally be general purpose, that is, not tailored to any particular knowledge representation.

To expand on the second of these ideas, we note that a knowledge representation specifies how a particular type of information is represented in a computer so that it can be used for reasoning [156, p. 16]. In the context of this thesis, the term refers to the variables that form the input to, and output of the musical decision maker, as well as that which is represented by those variables. Crucially, as will be highlighted in this section, the variables about which musical decisions are made are often not simple representations of traditional musical objects such as notes, chords or keys.
Instead they might be numerical parameters of digital effects units, probabilities to be used by lower-level generative processes, or other abstract quantities. Thus, a general purpose method for designing musical decision making behaviour is one that is not tailored to a specific knowledge representation.

In the following, we outline a formalism for understanding the computational structure of player-paradigm systems in which musical decision making mechanism is broken into three distinct parts. This is followed by an overview of the broad approaches that have been taken in the literature to designing player-paradigm systems. Three published systems are then described to illustrate the three-part formalism, before discussing the design of each part separately.

### 1.3.1 Computational Structure

Here, we outline a formalism for the computational structure of player-paradigm systems. It is a specific conceptualisation of the very general PQf model proposed by Blackwell [27, 29] as a framework for describing computer music systems. In the PQf model, the boundaries between the functionalities of the different elements and the time scales on which they operate are made deliberately nebulous. We present a more rigid version in which constraints are placed on the time scales on which certain elements operate, and in addition, certain features permitted by the PQf model are made more explicit. Then, in the following sections we show that many player-paradigm interactive systems can be fruitfully understood according to this less general model.

The PQf model describes two musical entities, A and B, (each of which may be human, silicon-based or otherwise) that are playing music in tandem. The sound output by A is denoted by X, and that output by B is denoted by Y. The sound X is modelled as a multi-stage, non-deterministic mapping from Y:

\[
Y \xrightarrow{p} p \xrightarrow{f(h)} q \xrightarrow{Q} X
\]  

(1.1)
The mapping $P$ maps the incoming audio signal, $Y$ to a feature vector, $p$. This feature vector is then used as the input to a mapping $f$ which may depend on a hidden state, $h$. The output of the mapping $f$ is a vector $q$, which is then mapped to an audio signal, $X$, by $Q$. None of the three mappings are necessarily deterministic, however it is stipulated that $P$ and $Q$ be ‘transparent enough for interacting humans to grasp and use during performance’ [27] (italics in original).

This model of interaction can be related to the models of the computational structure of interactive music systems presented by Rowe [152, pp. 10-23] and Winkler [181, p. 7]. Rowe’s model contains three modules: sensing, processing and response. Winkler divides the processing stage into three sub-modules: listening, interpretation and computer composition. Thus Winkler’s five-stage architecture is directly comparable to Blackwell’s $X \rightarrow P \rightarrow f \rightarrow Q \rightarrow Y$ formulation. In the following, to avoid these somewhat anthropomorphic terms, we will use the terms feature extractor, for the listening module which performs mapping $P$; decision maker, for the interpretation module which performs mapping $f$; and generator, for the computer composition module which performs mapping $Q$. These three elements are shown in Figure 1.1, of which more details are given below.

In our conceptualisation of the model, the feature extractor is deterministic and in addition, it may be adaptive. That is, $P$ is a function of an internal state, $h_P$, which may change with input $Y$. Furthermore, $Y$ can sometimes be a stream of MIDI data, rather than digital audio samples. In addition, the decision maker and generator are differentiated by the time scale at which they operate: the decision maker makes arrangement-level musical decisions, with its output parameterising...
the generator, which in turn generates individual notes and gestures (examples will
be given below). Finally, $Q$ may also include an internal state, denoted by $h_Q$, and $X$
may also be a stream of MIDI data rather than audio samples.

Before continuing, we note that frequently in this thesis we will use the term
musical agent to refer to an arrangement-level musical decision maker (i.e. the decision
maker in Figure 1.1). This derives from the correspondence between the decision
maker, situated between the feature extractor and the generator; and the artificially
intelligent agent, as defined by Russell and Norvig [156, Ch. 2], which receives
information about the environment from its sensors and acts on the environment
with its effectors. However, we acknowledge that it would also be valid to consider the
audio input (e.g. a microphone) to be the sensors, and the output (e.g. a loudspeaker)
to be its effectors, with the term agent referring to the trio of the feature extractor, the
decision maker and the generator.

Returning to the model, we highlight two features that are useful for under-
standing player-paradigm systems. The first is to allow $p$ to be a direct input to the
generator. For example, the most recent notes played by a musician might be used
directly in the generator to form a selection of pitches for use in its output. This is not
precluded by (1.1)—indeed Blackwell uses such a link in his Swarm Music system
[27]—but it is conceptually helpful to make this transfer of information explicit. The
use of the analysis data, $p$, to directly parameterise generative processes, $Q$, has been
referred to by Dean as extraction [67, p. 52]. Similarly, the incoming audio, $Y$, can be
used as a direct input to the generator. Again this is not precluded by (1.1)—being
also used in Blackwell’s Swarm Granulator system [27]—but it is useful to make the
link explicit. After Dean, we refer to the two direct links directly to the generator as
the extraction pathway.

In addition, we introduce two terms related to the model that will be used
throughout this thesis. First, we use the term music system variables to refer to $p$ and $q$,
above, that is, the musical information being supplied to the decision maker, as well
as that being output by the decision maker (input and output music system variables,
respectively). Second, we use the term *decision points* to refer to the (usually regularly spaced) points in time at which a musical agent re-evaluates the musical situation and sends instructions to the generator. In other words, it reads the values of the input music system variables, \( p \), and updates the values of the output music system variables, \( q \). Not all systems in the literature make musical decisions at discrete times like this, however, the systems arising from the original research described in this thesis do.

We refer to our version of the \( PQf \) model (as it is referred to in [25]) as the \( PQfe \) model (see Figure 1.1; the ‘e’ refers both to ‘explicit’ and to the extraction pathway). In the following, we will show how this model can be used to conceptualise a selection of published player-paradigm systems. However, first we discuss the approaches to designing such systems more broadly.

### 1.3.2 General Design Approaches

Broadly, there are three approaches to designing player-paradigm interactive music systems. Such a system can embody

(i) a model of musical behaviour;

(ii) a model of music itself or certain aspects thereof, which may be completely idiosyncratic, referred to in the following as a *structural model*; or

(iii) an abstract model whose inputs and outputs are mapped from and to the musical domain, respectively.

Of course these are not mutually exclusive and hybrid approaches are often taken. In particular, regardless of the primary decision making mechanism there is almost always a structural model, implicit or otherwise, embedded in the feature extractor or generator. Examples are given in the following.
1.3. The Design of Player-Paradigm Systems

**Behavioural models**

Behavioural models can arise from cognitive science, as is the case for instance, with Johnson-Laird’s jazz bass player [100]. More commonly, however, we see largely hand-crafted behavioural models such those in Hsu’s series of ‘timbre-aware’ systems [97–99], and others such as Lewis’ Voyager [116, 117]. Some models include anthropomorphic characteristics such as *boredom* [16, 99] and even *taciturnity*, as found in Free Improvisation System [57]. An alternative to hand-coding the parameters of a behavioural model is to use machine learning algorithms to derive them from recorded performances, as is done to create the virtual players for a ‘jam session system’ in [91].

A number of systems use similarity measures to chose musical material from a database. For example, in both [43] and [53], systems are described that continuously record a musician’s performance and then play back portions of the recording (subject to various manipulations) with the portions being chosen at a given time by their similarity to the signal coming from the musician. In these, systems, a (conceptually) simple behavioural model is implemented in which material is chosen that is maximally similar to the musician’s playing, according to some measure.

**Structural models**

Systems that generate musical material using transformative methods (i.e. by applying transformations to stored symbolic representations of music) could be seen to embody either a behavioural or a structural model. However, usually the transformations are standard ones from western music theory and so they can be regarded in the latter category. Examples include GenJam [17] in which jazz lines are generated by recombining phrases from a stored corpus and applying standard transformations such as reversals, retrogrades, inversions and repetitions; and a phrase generation process described in [183], whereby similar transformations are applied to phrases that have been played by the musician previously in the performance.

Other structural models can be arrived at by codifying music theory, by using
formal methods from probability theory or statistics, or by codifying idiosyncratic artistic or aesthetic ideas. Examples of the first approach include Rowe’s *Cypher* [151] and Blackwell’s *Swarm Music* [27], both of which derive the musical mode and tonic according to heuristic analyses of incoming notes. In contrast, the *Continuator* [136] and *Omax* [14] both use probabilistic methods to learn and generate musical phrases in the style of the musician with whom they are playing. Also in this latter category, the *GRI* system uses a model (trained offline using machine learning methods) to classify the playing ‘state’ of a musician, which it uses along with a probabilistic model, to choose its output [130].

Finally, with regard to the codification of particular artistic or aesthetic ideas, it is hard to imagine a system that is completely agnostic with respect to the musical preferences of its designer. Lewis has commented that ‘everyone’s machine expresses their aesthetic view’ [57, p. 23] and this holds even for systems intended to emulate the style of the musician with whom they are playing, such as *The Continuator*: it includes rules describing the relative importance of various musical attributes, for instance, the notion that pitch is more important duration (the *Continuator* is described in detail in Section 2.3.3).

**Abstract models**

The creation of systems from abstract models is an approach with which the *Live Algorithms* initiative strongly identifies [28]. Blackwell has designed a variety of systems in which the musician’s activity influences a simulated swarm of particles and the particles’ movement is then mapped back into the musical domain [24, 25, 27] or the audiovisual domain [26]. Another example is Di Scipio’s *AESI* which is based on a hand-crafted system of abstract and asynchronous mappings from input signals to output signals [69].
Hybrid models

Bown’s CTRNN and DT systems [30] are the result of a hybrid design approach: though they involve non-musical algorithms from the domain of machine learning (abstract models), the specific algorithm parameters were arrived at using evolutionary methods with musical knowledge embedded in the fitness function (structural model). Young’s NN Music [185] is another example. At its core are two artificial neural networks, one which attempts to learn the playing modes of the performer (a structural model) and another one which maps playing modes to output parameters (a behavioural model; more details are given below). A final example of is Casal’s Frank, which combines a similarity measure (behavioural model) with an evolutionary algorithm (abstract model).

1.3.3 Three Player-Paradigm Systems

In this section, we describe three published systems using the PQfe framework (see Figure 1.1). Each of these systems is highly regarded, having been used for multiple public performances with professional musicians. In particular, saxophonists John Butcher and Evan Parker have played with Hsu’s ARHS system; trombonist George Lewis has played (and made an album) with his own Voyager system; and a variety of contemporary improvisers have played with Young’s NN Music system.

ARHS

Hsu’s ARHS (Adaptive Real-time Hierarchical Self-monitoring) system [98, 99] is a ‘timbre-aware’ free improvisation system. It provides an example of a hand-coded behavioural model, though there are implicit structural models in the feature extractor and generator modules. With reference to Figure 1.2, the feature extractor derives categorical descriptions of loudness, tempo and timbre from the incoming audio at a rate of once per second, and from a sliding eight-second window over this feature data, derives a ‘performance summary’ which is an integer between 1 and 27
Figure 1.2: Hsu’s ARHS system.

describing the behaviour of the musician.

The performance summary forms the input to the decision maker where it is mapped via a probabilistic ‘adaptive performance map’ (the hidden internal state of the decision maker) to a ‘performance mode’, which parameterises each of a set of improvising modules (referred to by Hsu as ‘agents’) which together comprise the generator. Thus, the adaptive performance map operates at the time scale of arrangement-level musical decision making. The map is adaptive in that it uses the stability of the input state-output state combination (i.e. the duration for which it lasts) to adjust the probability of that mapping recurring. Short-lived combinations are assigned lower probabilities; it is assumed that the human improviser changed mode because the mapping was inappropriate.

The modules which comprise the generator are generative processes which control virtual instruments and are parameterised by the performance mode. They also receive ‘parameter curves’ (which describe the variation of input timbral characteristics over time) from the feature extractor via the extraction pathway. These are used both in sound generation and to derive ‘potential trigger events’ which are temporal events that can trigger instantaneous output changes, intended to
give the system a short-term responsiveness to the musician’s actions. Pitches are chosen by analysing the incoming audio for the occasional stable pitch and applying ‘simple transformations’. Finally, the behaviour of each ‘agent’ is also governed by various probabilistic rules, such as the possibility that out of ‘boredom’ it will change performance mode.

**Voyager**

Lewis’ Voyager [116, 117] can be characterised as a generative system with parameters that can be influenced in real-time by the performance of an improvisor. In terms of the PQfe model, the feature extractor takes the musician’s MIDI data as input and calculates a set of features representing the instantaneous state of the input, in addition to averages calculated over a time window (see Figure 1.3). Lewis describes this as a representation of ‘a sonic environment ... within which musical actions occur’ [116]. Examples of features include volume, velocity, sounding duration, inter-onset interval, register and interval width.

The decision maker takes the ‘state’ describing the musician’s playing as input in order to parameterise the generator, which comprises a set of virtual players—a
‘locally intelligent orchestra’ [116]—each of which can output its own stream of MIDI notes. The parameters include ‘ensemble combinations’ by which the virtual players are combined and recombined into subgroups, each of which in turn is parameterised by a ‘behaviour specification’ which includes a choice of melody algorithm (of which there are 15), ‘pitchset’ (of which there are 150), timbre, tempo, ornamentation and other parameters. New ensembles are created, and old ones destroyed at irregular intervals generally between five and seven seconds in length (this range can be set on a per-performance basis). Thus, as with the ARHS system, the decision maker operates on the time scale of arrangement-level decision making to parameterise a generator which operates on the time scale of sound objects (i.e. individual musical notes).

**NN Music**

Young’s NN Music is a ‘performer-machine’ system that has had a number of incarnations which can be differentiated primarily by the generators used [185]. Described in terms of the PQfe model, the feature extractor analyses incoming audio in 50 ms windows and derives pitch information as well as a set of perceptually motivated descriptors such as loudness, brightness and duration between events (see Figure 1.4). From these, a state vector is calculated comprising the normalised mean and standard deviation of the various descriptors, measured over a time window of between five and ten seconds in length. While the state vector is recalculated every 50 ms, it is this window that determines the time scale on which change can occur, i.e. that of arrangement-level musical decision making. This state vector forms the input to the decision maker.

The decision maker is a four-part system comprising (i) an artificial neural network (ANN), which we will refer to as $A$, which maps the state vector to a length-$n$ vector of classification weights (see below); (ii) a ‘power function’ which acts to increase the high-valued classification weights and decrease the lower ones; (iii) a ‘covert mapping’ which randomly reorders the classification weight vector from time
to time; and finally, (iv) a second ANN, $B$, which maps a length-$n$ vector to a length-$m$ vector that parameterises the generator. The purpose of ANN $A$ is to classify the musician’s performance, however, it is not created (i.e. trained) in advance. As new input states arrive during the performance, some are used as prototypes, meaning they are given a classification label and ANN $A$ is re-trained with $n = n + 1$ outputs to accommodate the new class. A state is chosen as a prototype if it is sufficiently different from all previous prototype states, according to an idiosyncratic ‘fitness test’. The purpose of ANN $B$ is to map the musician’s performance classification to a parameterisation of the generator. ANN $B$ is pre-trained to produce particular outputs for a selection of ‘ideal’ input conditions—presumably those corresponding to classification weight vectors in which a single element has a value of unity and the others are all zero.

A variety of generator modules have been used in different versions of the system. Thus, the length-$m$ vector that parameterises the generation has been used in different ways. However, in general it specifies probability distributions governing the details of the musical output. Finally, the choice of pitches used by the generator is derived using a procedure based on the chord multiplication scheme due to Boulez.
20

Chapter 1. Introduction

(see, e.g. [93]).

1.3.4 Common Design Strategies

In the above, we have presented three published player-paradigm interactive music systems according to the framework provided by the PQfe model. In particular, in each case there was a decision maker operating at the time scale of arrangement-level musical decision making to parameterise a generator operating on the sound object time scale. Furthermore, a wide variety of musical representations were used as the input to, and output from, the decision makers. These included abstract ‘state’ variables, timbral descriptors and probability distributions. In the following, we support this further by presenting a survey of the design approaches that have been used for each of the three modules (feature extractor, decision maker and generator) individually. As well as highlighting the variety of representations of musical information, and the use of arrangement-level musical decision making this will provide further context for this research.

**Feature Extractors**

Feature extractors may process either MIDI (symbolic) or audio (sub-symbolic) signals. These are treated separately in the following.

- **Symbolic Representations:** In earlier systems, feature extractors dealt exclusively with symbolic data usually arriving from a MIDI instrument or a pitch to MIDI converter. Features derived commonly included sets of pitches being used by the performer and higher level descriptors relating to rhythm and harmony as well as low-order statistics of various related quantities. Rowe’s Cypher included a complex feature extractor deriving note density (i.e. the rate of onsets) and information related to register and dynamics, as well as harmonic descriptors, beat-tracking information and the locations in time of phrase boundaries. In addition, a set statistics-based calculations were used to classify some of these quantities as varying with or without ‘regularity’ [151]. Voyager (see above) also made use of statistics derived from MIDI
data; as Lewis notes, ‘My big thing is averages ... Then you can map all of that to the output’ [49]. Blackwell’s Swarm Music and Swarm Granulator [27] is a more recent system to use simple analyses of MIDI data as input to the decision maker. Another approach to symbolic feature extraction is that used in Beyls’ Oscar, which monitored the incoming MIDI stream for particular pre-programmed pitch patterns [16]. In a similar vein, there a number of systems which extract phrase or motivic information including Free Improvisation System [57] and others, previously mentioned [14, 17, 136, 183].

- Sub-Symbolic Representations: Many more recent systems take advantage of increased computational power to derive a much richer description of the incoming audio signal. In addition to information relating to pitch and rhythm, descriptors include amplitude envelopes and a variety of perceptually informed quantities such as loudness, spectral brightness and noisiness [43, 53], as well as mel frequency cepstral coefficients [46]. Other signal processing analyses have also been used, such as the power spectrum [183]. Frequently, statistics of these quantities, such as the mean and standard deviation are calculated for a constant time window [185], or coupled with a phrase detector, on a per phrase basis [43, 53]. Gestural data has also been derived from the time-varying quantities such as those mentioned above to be used in the control of synthesis and effects parameters [69, 99] in the manner of adaptive effects [175] (i.e. effects in which the parameters change according to the characteristics of the incoming signal).

**Decision Makers**

As outlined above, in our specific conceptualisation of Blackwell’s PQf model (which we refer to as the PQfe model), the decision maker is an arrangement-level decision maker operating on a time scale of the order of a few seconds. This means that the generator, for which design approaches are outlined below, can also be said to carry out ‘decision making’ (since it is responsible for generating specific musical phrases and patterns), but on the sound object time scale. However, our contention is
that there is, in these systems, a clear distinction between the two types of decision making. We support this in the following with further examples of systems featuring explicit arrangement-level musical decision making. In addition, we identify two other types of musical change commonly modelled by the arrangement-level musical decision maker of interactive player-paradigm systems: change on the macro time scale, and abrupt, sporadic change.

- **Arrangement-level decision making:** As in NN Music, Voyager and ARHS (see above), it is common to use a process (not necessarily deterministic or transparent\(^3\)) that operates on the time scale of arrangement-level decision making and maps from an input state to an output state. Many systems already mentioned are examples of this, including Rowe’s Cypher in which the ‘level 2’ mapping occurs over ‘phrase-length spans of time’ [151], and GRI, which operates at least on the time scale of musical phrases in order to classify the playing ‘state’ of the musician [130]. Similarly, Carey’s _derivations_ selects new musical material from a database each time the musician finishes a phrase [43]. Other systems involve algorithms with internal dynamics that change much more quickly, but the way that their output is filtered or sampled means that decision making is occurring effectively at the time scale of arrangement-level musical decision making. These include Blackwell’s Swarm systems [27] and Bown’s CTRNN and DT systems [30].

- **Change on the macro time scale:** Another frequent characteristic of decision makers is a mechanism giving rise to gradual change that occurs over the duration of the performance (i.e. the macro time scale; see above). An early example of this is Beyls’ Oscar [16] in which the lower-level processes are influenced by a slowly changing internal two-dimensional state related to notions of ‘boredom’ and ‘stimulation.’ In addition, as described above, the mapping component Hsu’s ARHS system is adaptive, changing slowly over the course of the performance. The slow change can also arise from the algorithm used, as in Bown’s CTRNNs which are reported to ‘have an elegant drifting behaviour’ [30]. Other examples of this include

\(^3\)We use the term transparent to refer to a process that would be readily understood by a musician playing with the system.
systems that involve real-time genetic algorithms optimising populations according continuously changing fitness functions [46, 183], and in these, slow change is provided by the evolution process itself. Finally, slow change may be introduced by the use of a database which grows as new material is added from the musician’s performance. This occurs in _derivations [43] which continuously updates a database of audio material and Free Improvisation System [57] which accumulates motifs over the course of a performance.

- ‘Surprise’ mechanisms: Finally, many systems include surprise mechanisms. We introduce this term to refer to a process which will, from time to time, abruptly and unexpectedly change the output of the system. An example is NN Music (see above) which performs a random re-configuration of the classification weight vector (a ‘covert mapping’) to change the responses of the system. Additionally, in Carey’s _derivations, there is a mechanism to arbitrarily break the monotony when it detects that a single database phrase is being used for too long [43]. Another example is the inclusion of probabilistic rules in ARHS (see above) such as the possibility that out of ‘boredom’, the system will change its performance mode [99]. In a similar vein, Collins’ Free Improvisation System involves high-level behavioural parameters such as ‘taciturnity’, ‘shyness’ and ‘insularity’. These are reconfigured during the performance ‘to cause … interest, in the spirit of musically varied behaviour’ [57]. Finally, in the case of Casal’s Frank, it is interesting to note that the addition of a higher level process to automate the breeding frequency and surprise parameters (two parameters of the system), potentially changing the rate of change in the system from time to time, lead to improved engagement [45].

Generators

Of critical importance to the aesthetic characteristics of a player-paradigm system is its choice of musical material to output. This is largely determined by the design of the generator. We have noted the views of Young and Bown that the design of this component is that most similar to standard music composition. Relevant to this is the
fact that in many publications, the feature extractor and decision maker are described in considerable technical detail, but the generator is not, indicating a widespread view that the generator is the result of an artistic or compositional process more than an engineering or technical one\(^4\). Nevertheless, despite the frequent lack of detail with regard to the generator, some common strategies can be gleaned from the literature.

- **Pitched Material:** Where the generator outputs pitched material, there are a number of approaches to choosing the pitches to use. In some cases the pitches are those used by the musician subject to various (often unspecified) simple transformations, as in *ARHS* [99] or more sophisticated ones, such as the aforementioned chord multiplications performed in *NN Music* [185]. Alternatively, output may be constrained to notes from a particular musical key or mode [27, 183]. Also, as previously mentioned, to generate sequences of notes, it is common to apply musically informed transformations to those played by the musician [17, 151, 183].

- **Audio Material:** As previously noted, there are a number of examples of systems that choose audio material from a database according to its similarity to, or dissimilarity from, the audio arriving from the musician. In some cases the retrieved audio is subject to significant manipulation, before it is output [43, 53]. In a system published by Yee-King [183], a genetic algorithm runs in real-time with the aim of evolving synthesized timbres that are close to the timbres being produced by the musician, thus the difference between the synthesized tones and those used by the musician arise from the dynamics of the genetic algorithm and the limitations of the synthesis method used. *Frank* demonstrates another take on the similarity-based approach, where the musician’s audio is used to influence an evolutionary process and the results of that process are used to query the audio database [46].

- **Short-term responsiveness:** Hsu espouses an approach to designing generators that includes ‘short-term responsiveness’ to individual sound objects [99].

\(^4\)The issue of reproducibility in publications of this kind, and how it relates to the tension between the artistic and scientific disciplines, is one we have raised in the ‘musical metacreation’ community [33].
1.3. The Design of Player-Paradigm Systems

mentioned above, he achieves this through the identification of potential trigger events in the incoming audio, which may be responded to immediately. This idea is not new, for instance Beyls’ Oscar [16] exhibits this by allowing, under some circumstances, a ‘reflex response’ to bypass its longer term generative processes. Finally, short-term responsiveness is another behavioural feature that can arise from the algorithm used, rather than being due to an explicit design decision: In Bown’s DT-based system it was due in part to the categorical nature of the DT output which would often change suddenly in response to an abrupt change in the input [30].

1.3.5 Summary

We have presented a survey of player-paradigm interactive music systems, focussing on the approaches taken to their design. We have noted a number of prevalent design strategies of which two are particularly relevant to the research in this thesis. First, systems frequently operate explicitly on the time scale of arrangement-level musical decision making and the sound object time scale and moreover, the decision making on the former time scale is usually independent of that on the latter one, that is, the output of the generator does not affect future decisions of the decision maker. This supports the notion that it is reasonable to consider arrangement-level musical decision making separately from that which occurs on the shorter time scale. Second, the decision maker is used with input and output variables representing a wide variety of musical concepts, and thus any method for designing such components is required to be general purpose, that is, to be agnostic with respect to the knowledge representation used.

Finally, we note that there are certain interactive music systems that cannot be usefully conceptualised according to the PQfe model. An example is the Continuator [136], which does not have an arrangement-level musical decision maker. Its output is simply determined by a generator which generates musical phrases, one at a time (details of this are given in the context of musical modelling in the next chapter). There is no higher-level organisation except that imposed by the musician
with whom the system is playing. However, the Continuator, unlike the majority of the systems mentioned above, was not designed to be a fully-fledged participant in contemporary experimental music. Instead it was a study of techniques for the combining interactivity and style imitation [136] and therefore, was not subject to the same concerns.

1.4 End User Tools for Designing Musical Decision Making

There exist a number of software tools, designed for non-programming end users, for creating interactive and generative systems. Tools can be divided into three categories, according to the types of design activities undertaken by the user. Users of tools in the first category essentially create a model of music using musical concepts and abstractions thereof. Typically, design activities include specifying rules and probabilities that govern the generative processes. This can be related to the structural modelling approach to designing player-paradigm music systems. Tools in the second category employ extra-musical algorithms combined with schemes by which to map generated data to musical events. A system is created by configuring the algorithms and the relevant mappings. This can be related to the use of abstract models for designing player-paradigm systems. Tools in the final category allow compositions to be generated according to very high-level descriptors chosen by a user.

1.4.1 Tools for Designing Structural Models

Exemplars of the first category include software such as M and Jam Factory [187]. M allows a user to define a set of musical patterns which are used as source material for a set of probabilistic and cyclic transformative processes that can re-order notes and alter their dynamics, articulation and rhythm. Jam Factory also employs a probabilistic approach: it derives Markov models from sequences of pitches and
durations in recorded MIDI data and uses them to synthesise new material.

Noatikl\(^5\) is a more recent work in this category. The user defines an ensemble of voices, each of which generates a monophonic stream of MIDI notes and can be configured to do so according to a pre-defined pattern, or according to a set of user-selected probabilistic rules. For example, the user can define a discrete probability distribution from which each pitch will be drawn and another distribution from which each duration will be drawn. However, the system also employs certain global heuristics which may alter these distributions under some circumstances; for instance, note durations may be selected so that the voice does not sound across a bar line. Voices have a multitude of user-selectable attributes such as the length of phrases and the gaps between phrases, the voice type (e.g. rhythmic, repeating or following another voice), and many others, some type-specific. Noatikl can also respond to incoming MIDI data in various ways. Brown and Kerr [35] review related systems in this first category, all of which are designed specifically for manipulating the objects of western tonal music (notes and chords, etc.).

It is also worth noting that certain general purpose music production environments offer rudimentary features for designing musical decision making. For example, Ableton Live (see next chapter) allows a user to define rudimentary probabilistic rules describing what happens after a clip (a short segment of musical material) has been played back. This amounts to the specification of a first order Markov model (see next chapter) for autonomously generating clip sequences from a limited set of options (arbitrary model parameters are not permitted).

\[1.4.2 \text{ Tools for Designing Abstract Models}\]

Tools in the second category frequently employ spatial metaphors. IanniX [55] was inspired by Xenakis’ UPIC system (see, e.g. [172]). In the terminology of the software, a score is a 3D space containing various elements; the user defines trajectories along

\(^5\)See: intermorph.com/noatikl.

\(^6\)A precursor of Noatikl is SSEYO Koan with which Brian Eno composed his collection of generative music works, Generative Music I.
which cursors travel at specified speeds, as well as various types of static objects. Both objects and cursors have user-defined finite spatial extents and when a cursor intersects with an object during its travel, a discrete control signal is created which can be used to trigger any multimedia event. A related work is Nodal [127] in which a user creates a network of nodes connected by edges. Each node is associated with a note or group of notes and multiple edges can lead to or from a given node. One or more voices travel around the network, emitting the MIDI notes specified by the nodes through which they pass. Similar to IanniX, the length of an edge governs the time between note events. Cyclical and indeterminate paths can give rise to emergent musical properties and long-term structure.

Another metaphor commonly used is that of the physical world. Bouncing balls abound in software for the iOS platform, such as the Soundrop which allows a user to draw a set of lines on the screen of a mobile device. A stream of small graphical balls then ‘drop’ from the top of the screen and bounce off any lines that they contact. Each time a ball hits a line, a note is played with its pitch depending on the length of the line. In addition, the Lemur software for creating custom touchscreen interfaces to control music software includes ‘physical modelling’ interface widgets along with a standard selection of knobs and sliders. For example a graphical ball can be set bouncing around a box with control messages being sent to trigger multimedia events each time the ball hits a side. Plug-ins such as the Nova 3 generative sequencer introduce similar functionality to standard music production platforms.

1.4.3 Generative Tools Configured by High-Level Descriptors

Wolfram Tones allows a user to select high-level descriptors (e.g. genre, instrumentation, pitches, tempo and duration) to generate a composition. Underlying the software are algorithms based on 1-dimensional cellular automata (CA) [182]. The CA parameters are also user-selected, however, we still place this in the third category because they are completely opaque: a user can experiment with them, but

---

7See: tones.wolfram.com.
the only deliberate steps that can be taken (i.e. steps with expectations of particular results) are related to the high-level descriptors.

*Songsmith*\(^8\) and *Band-in-a-Box*\(^9\) also allow compositions to be created according to high-level descriptors. In the case of the *Songsmith*, a backing-track is generated in a given style, to accompany a vocal part recorded by the user. *Band-in-a-Box* allows backing tracks to be created according to a user-specified harmonic structure, again with options related to style, tempo and other high-level descriptors.

### 1.4.4 Conclusion

Of the programs mentioned above, many are quite idiosyncratic; some are as much interactive art works as they are tools; others are closer to being controllers for music software, albeit ones that break the ‘one-gesture-to-one-acoustic-event’ paradigm \([180]\); and finally, others support only a very limited range of outcomes. Those that were created for musicians (rather than casual users) include *Noatikl, M* and *Nodal (Jam Factory is no longer available)* and these are intended exclusively for generating (and receiving) streams of MIDI notes (i.e. they are specific to the concepts of traditional western music notation). In addition they are primarily for creating generative compositions, with few features for introducing interactivity. In the next section, we clarify this last statement by giving a brief overview of the concept of *interactivity* as it is understood in the field of computer music.

### 1.5 Interactivity

In the interactive computer music literature, considerable attention is paid to the distinction between systems which are *interactive* and those that are merely *reactive*, yet a clear definition of interactivity in the context of computer music systems has not been proposed. Rafaeli [146] gives a recursive definition of interactivity in turn-based communication as ‘the extent that in a given series of communication

---


exchanges, any third (or later) transmission (or message) is related to the degree to which previous exchanges referred to even earlier transmissions.’ He contrasts this with a definition of reactive communication in which a message from system A to system B only depends on the previous message from system B to system A, but nothing before that.

In English, the term interactive simply refers to mutual influence, or ‘two-way flow of information’ [128], and it is hard to imagine any musically relevant device or system being non-interactive in this sense\textsuperscript{10}. However, in the computer music literature, the key to the distinction between reactivity and interactivity is the perception of the human actor. According to Drummond [71], ‘A system consistently providing precise and predictable interpretation of gesture to sound would most likely be perceived as reactive rather than interactive.’ Thus, a traditional musical instrument, because it is perceived to respond to stimuli in a mechanistic and predictable way, is reactive, rather than interactive, despite the clear presence of a two-way flow of information between the player and the instrument.

This span between reactivity and interactivity can be related to Dennett’s three stances from which one may view the behaviour of an object [68]. The physical stance is the lowest level of abstraction at which behaviour might be viewed. This would apply, for example, to an acoustic musical instrument. The design stance is the intermediate level of abstraction and it may apply for instance, in the case of an audio effect. At the highest level of abstraction is the intentional stance from which one is concerned with, for example, the ‘beliefs’ and ‘intentions’ of the object.

The intentional stance clearly corresponds to Paine’s idea of interactivity as he writes ‘the term interactivity is therefore widely abused . . . most systems are not interactive, but simply reactive or responsive because they lack a level of cognition’ [139]. That the ‘level of cognition’ of a computational system is a perceptual attribute, is reinforced by Jordà [102], who links interactivity to the potential of a system for producing unexpected results, whether through ‘non-linearity, randomness

\textsuperscript{10}In the broader computer science literature this definition holds too, since interactive software is simply software that accepts and responds to input from users [166].
or ungraspable complexity’; a key attribute of an interactive system is the extent to which a human finds it unpredictable, rather than the extent to which it actually is unpredictable. Thus, a high ‘level of cognition’ could be attributed to a complex, but completely deterministic system, for instance, simply because it is difficult to predict its behaviour.

For the purposes of this thesis we shall be consistent with the computer music literature cited above and consider an interactive music system to be one that is influenced by the activities of another musician (i.e. the data arriving at its inputs) and whose behaviour might be expected to elicit the attribution to it of some cognitive ability. In other words, an interactive music system is one which successfully strikes a balance between being simply reactive at one extreme and behaving so as to make the connection between input and output imperceptible at the other [71]. Were this definition to fail, we would defer to Rafaeli’s acknowledgement that ‘The common feeling is that interactivity, like news, is something that you know when you see it’ [146]. In general, music performance software is tending towards increasing demand for interactivity with increasing ‘levels of cognition’ that can mimic both a human’s musical understanding, and decision making.

### 1.6 Aims of this Work

At this point, we have surveyed player-paradigm interactive music systems that have been developed in the field of computer music research in order to show that it is reasonable to consider arrangement-level musical decision making separately from lower-level decision making, and also to motivate the requirement that methods for designing the former be general purpose. We have provided additional motivation for developing tools for designing musical agents, with reference to the Ableton Live software and the performance paradigm that it supports (more details are given in the next chapter). Finally, we have provided an overview of currently available tools for designing real-time musical decision making systems (primarily generative ones)
and highlighted the extent to which they are specific to the concepts of traditional western music notation. For clarity and reference in later chapters, we now list a set of requirements for methods to support end user design of arrangement-level musical decision making (musical agents, in our terminology).

### 1.6.1 Requirements for methods for designing musical agents

We seek to develop methods for designing arrangement-level musical decision making that are:

1. **General purpose:** to the extent possible, not specific to particular knowledge representations, that is, specific types of musical information (see Section 1.3).

2. **Generative and interactive:** suitable for designing real-time generative or interactive behaviour.

3. **Suitable for non-programming end users:** accessible to users with no expertise in conventional computer programming (i.e. the use of traditional computer programming languages).

4. **Real-time:** suitable for real-time computation.

With regard to the third of these requirements, we note that ideally, a method would not require any expertise outside of the musical domain (i.e. music performance, music theory and use of music software or hardware). However, while this may be aspired to, it may not be feasible. In the following, we acknowledge that no single method could possibly support the design of all conceivable musical agents, and suggest a more reasonable aim with regard to the range of contexts in which such a method would be useful.

To begin, concerning research in artificially intelligent music systems, Bown [30] contrasts *prescribed* and *unprescribed* behaviour types. Prescribed behaviours are those corresponding to established styles and can be judged against representative examples of that style. Conversely, unprescribed behaviours are those that do not
correspond to particular styles. Interactive or generative music systems exhibiting
behaviours of the former type are usually created in the context of scientific or
engineering research, whereas those exhibiting behaviours of the latter type usually
result from creative projects undertaken at least partly with the aim of generating
new musical ideas.

Related to this is Perkis’ differentiation between ‘two responses to complexity,
two approaches to the problem of design’, one being the “'crafting’ school’ who
‘get what they want’ and the other being the more experimental “'wild system”
school’ who ‘want what they get’ [36]. While Perkis was referring to the use of
computers to make music in general, these two design approaches can considered in
relation to the design of musical decision making. Members of the Live Algorithms
community [28] can be seen as adherents of the ‘wild system’ school, interested
in exploring the dynamical behaviours of algorithmic processes in the context of
experimental music. Clearly there can be no single method to support the design
of this type of system, since such a method would have to be capable of modelling
arbitrary dynamical behaviours, from recurrent neural networks to cellular automata.
The same argument could be made with respect to design tools for ‘crafters’ since
such tools would also have to be capable of modelling any conceivable behaviour.
However, at the outset of this research, it seemed more reasonable to expect that
methods could be found that would support the design of arrangement-level musical
behaviour that would be of utility to a significant proportion of musicians aiming to
develop interactive and generative systems for their creative practise.

To conclude, we note that the general purpose requirement is not necessarily re-
lated to the range of musical contexts to which a given design method might be
applicable. The systems surveyed in Section 1.3 involved many different know-
ledge representations (thus motivating the need for a general purpose method) and
yet they were overwhelmingly intended for a single musical context, that of free
improvisation in the tradition of contemporary art music.
1.6.2 Research Questions

With our overarching goals outlined, we now give details of the specific research questions addressed in this thesis. They are listed below in the form of objectives, with the relevance briefly indicated in each case.

I To investigate the affordances of the partially observable Markov decision process (POMDP) as a framework for designing musical agents with respect to the requirements listed in Section 1.6.1: The POMDP is a mathematical model of interaction but its use in the field of interactive computer music has not been studied. This research aimed to shed light on the affordances of the POMDP to a non-programming end user. Specifically, we identified potential applications of the model, as well as challenges that must be addressed with further research.

II To design, implement and evaluate a general-purpose, history-based system for modelling arrangement-level musical performance, based on the similarity measure for sequential patterns: A number of well-known interactive music systems (such as the Continuator; see Section 2.3.3) use history-based models of musical performance. That is, the user specifies the behaviour of the system by providing a training data set which is then used to make musical decisions. However, the systems in the literature have been based on procedures for searching the example data that are specific to musical note sequences. This research aimed to evaluate one method for adapting history-based models to arrangement-level musical decision making with respect to the requirements outlined in Section 1.6.1. The evaluation accounted for arguably the two most important characteristics of a history-based system: the quality of musical decisions made and the size of the training data set required.

III To design a system for designing musical agents that retains the benefits of programming by example and satisfies the requirements listed in Section 1.6.1, while removing the need for large amounts of training data: Our
work concerning question II indicated that systems based purely on programming by example are likely in general to require too much training data from the user. This research aimed to find ways to mitigate this and, in the context of this thesis, can be broken into the following sub-goals:

(i) **To identify the machine learning techniques to underly the system:** This research aims to identify a machine learning algorithm, or combination of algorithms that can be used to train models of arrangement-level musical decision making from training data sets, in accordance with the requirements listed in Section 1.6.1 (particulaly *general-purpose* and *generative and interactive*). This problem has not been previously addressed in computer music research. The evaluation of the candidate system is addressed in research questions III-(iii), III-(iv) and III-(v).

(ii) **To implement a real-time decision maker that can use the learnt models to take part in musical performance:** This is relevant to the *real-time* requirement listed in Section 1.6.1. The candidate system designed to address III-(i) used *constraint-based* model of musical performance. Such models are, in general, not amenable to real-time use. This research aimed to implement a system for using such models subject to real-time constraints. The evaluation this system involved measuring its performance with realistic models.

(iii) **To evaluate the modelling capabilities of the system:** This is related to the requirement that the system be useful to as wide a variety as possible of musicians aiming to develop interactive and generative systems for their creative practise (see Section 1.6.1). Research questions III-(i) and III-(ii) resulted in a complete system for learning and performing arrangement-level musical decision making. This research aimed to evaluate the extent to which this system could be used to learn and reproduce real musical data.
(iv) To evaluate the system with respect to incorporating it into the workflow of contemporary computer music practitioners: This research aimed to find ways in which the system might restrict a practitioners workflow, as well as features that make it more or less easy to apply in different musical contexts. Since the case studies involved designing a variety of interactive and generative systems using different knowledge representations, the outcomes of this research are relevant to the generative and interactive and general purpose requirements (see Section 1.6.1).

(v) To assess the usability of the system by non-programming end users: This research aimed to identify issues with the usability of the system both through theoretical analysis and practical study. As such, it is related to the non-programming end user requirement of Section 1.6.1.

These questions will be further contextualised in an outline of the remaining chapters given below.

1.7 Outline and Contributions

The remainder of this thesis is structured as follows:

Chapter 2: Background

An overview is given of the Ableton Live and Max music platforms. Basic terminology and concepts are introduced, related to Machine Learning and Markov models. In particular, we introduce supervised and unsupervised machine learning, and outline the typical workflow associated with machine learning, as well as that associated with an alternative approach more suited to users with no machine learning expertise, known as interactive machine learning.
Chapter 3: A Study of the Potential of the Partially Observable Markov Decision Process for Designing Musical Agents

We study the partially observable Markov decision process (POMDP) as a potential framework for designing musical agents. This is an elegant model that takes into account certain fundamental aspects of interaction: the values of possible outcomes, predictions of the consequences of possible actions, and uncertain knowledge about the state of the world. We show its potential as a means to design the responses of player-paradigm interactive music systems (i.e. how they react to particular stimuli). However, beyond systems that are simply reactive, its use in other fields has shown that the POMDP model can give rise to highly sophisticated behaviours, and this regard, its promise is not fulfilled. It remains unclear what knowledge representations would support the design of behaviours relevant to music, and in addition—if such representations were found—what strategies would lead to effective reward functions (a particular parameter of the model).

Chapter 4: Programming Agent Behaviour by Example

We turn to programming by example to infer a model from examples provided by a musician. We present a novel instance-based machine learning algorithm that can be used as a general purpose method for the design of arrangement-level musical decision making. We show the promise of this straightforward approach, but demonstrate that it is likely, in general, to require too much training data for a musician to supply. Furthermore, we highlight a more general issue with this paradigm which is that a model resulting from using machine learning algorithms as a black boxes is opaque and inflexible; it cannot be modified by the user.

Chapter 5: The Agent Designer Toolkit

We identify an alternative approach, still based on programming by example, but which allows the musician to compensate for a paucity of training data by input-
ting musical knowledge into the system. The software developed to support this approach is the Agent Designer Toolkit which in part supports an interactive machine learning approach to programming by example, but which requires the user to perform feature selection and configure other learning parameters (we refer to these activities collectively as designing the learning configuration). In addition to being general purpose, this software clearly meets our aim to support the design of musical agents by musicians with no expertise in conventional computer programming. However, we posit that a requirement to engage in the analytical process of designing the learning configuration would curtail its affordances to some musicians.

Chapter 6: Implementation of a Real-Time Musical Decision Maker

The implementation of the Agent Designer Toolkit is described. This includes perhaps the first constraint-based musical decision maker that can be run with the strict bounds on execution time that are required for real-time operation. It takes advantage of the properties of the binary decision diagram data structure.

Chapter 7: Modelling Arrangement-Level Musical Decision Making with the ADTK

We describe an empirical study to characterise the modelling capabilities of the Agent Designer Toolkit, and also develop design techniques with the aim of informing the design of a higher-level interface that would mitigate the requirement for musicians to engage in designing the learning configuration. The key results from this chapter are the characterisation of the software’s modelling capabilities, as demonstrated by the musical agents that were produced, and the identification of important techniques for using the software to capture various common types of musical patterns. In addition, we identify a variety of ways to improve the modelling possibilities.
Chapter 8: Case Studies: Agents Designed

We take a more holistic view of the agent design process, and how it is affected by the wider context which involves working with musicians subject to logistic constraints imposed by collaborative design scenarios. We illustrate the variety of contexts in which the Agent Designer Toolkit can be used, and the speed, probably unprecedented, with which it can be used to develop prototype musical agents from pre-prepared sets of musical materials. In addition, this study provides for a more nuanced understanding of the difficulties with the analytical process of designing a learning configuration.

Chapter 9: A Study of the ADTK from the Perspective of End-User Programming

We present a two-part usability study of the ADTK. In the first part, the Cognitive Dimensions of Notations, a theoretical framework from the human-computer interaction literature, is used to analyse the usability characteristics of the ADTK. We then present the results of a preliminary usability study in which a participant was asked to design a musical agent to emulate the style exhibited in a set of example performances. We identify a range of ways to improve the usability of the ADTK, as well as future research challenges.

Chapter 10: Discussion

We draw together the results from the two studies of the Agent Designer Toolkit, to characterise its current potential as a creativity support tool, and how the results of the studies might be used to ameliorate its shortcomings through (i) enhancing its modelling capabilities, and (ii) providing a higher-level user interface. Central to the latter is the following idea: now that a method has been devised whereby a user can input musical knowledge to compensate for a lack of training data, it should be augmented with a range of pre-programmed musical knowledge that can
be simply selected according to a user’s description of the musical examples. We make a number of concrete proposals for augmentations of the software and list future HCI studies that will aim to further research exactly what pre-programmed musical knowledge should be included and how it should be presented.

Chapter 11: Conclusion

The findings of this thesis are summarised.

1.7.1 Contributions

Following is a succinct list of the contributions of this thesis.

- An evaluation of the potential of partially observable Markov decision processes as algorithms for real-time musical decision making
- The development and evaluation of the similarity algorithm which supports designing arrangement-level musical decision making by example
- The development of a new class of models for musical decision making, which comprise association rules, variable-order Markov models other constraints
- The implementation of The Agent Designer Toolkit: software for designing the aforementioned models for musical decision making
- A method for solving constraint-based musical decisions subject to real-time constraints based on binary decision diagrams
- A comprehensive characterisation of the aforementioned models with respect to their potential to capture the musical styles of practising electronic musicians
- An autoethnographic study of using the Agent Designer Toolkit in the collaborative development of interactive and generative music systems
- A usability study of the Agent Designer Toolkit
• A user-centred design study of the requirements of stakeholders in collaborative scenarios involving musical agents

• An evaluation of the potential of the Agent Designer Toolkit as a creativity support tool

• A set of proposals to improve the Agent Designer Toolkit as a tool for interactive unsupervised machine learning

1.8 Personal Statement

I was drawn to this topic through my practise of composing music systems using Max. This practise was rooted in a fascination with the idea that you can build systems that generate music in real time, entirely from the ground up, with processes operating on every time scale from samples to minutes. I would generally start by building individual synthesizers, sample manipulators and audio effects and then construct sequences or pattern generators to play them. I found that it came fairly naturally to me to build small probabilistic drum rhythm generators and melody generators. These were often based on simple low-order Markov processes or ad-hoc heuristic schemes, all of which could be implemented quite readily in Max. However, I found that developing probabilistic processes to control the output on a longer time scale, in a way that made musical sense, was much more difficult.

At some point I found a particular Max patch online that was simultaneously extraordinarily exciting and very frustrating. It had purportedly been stolen from the computer of one of the members of the electronic duo Autechre, and distributed online (or else it was a masterful feat of reverse engineering and I was duped, but the results are the same, either way). I had misgivings about the origins of the patch but it never occurred to me not to download and open it. At the click of a button, this single Max patch generated endless variations of the composition Liccflii, from Autechre’s record, EP7, and it was absolutely enchanting to watch it do so. The synthesizers were implemented from scratch, as well as the probabilistic sequencing
elements (off-kilter and lovely, as Autechre’s tend to be), but most exciting to me, it exhibited convincing long term musical structure.

What was frustrating was that the generative engine of the patch was a tangled web of random number generators and counters controlling the flow of data through a veritable maze of gates and switches. While it could be decoded, and I spent a long time doing so, it was very difficult to learn from. There was some evidence of specific methods underlying its construction. For instance, two cyclical counters would operate in parallel, incrementing their values once every eighth note, with one counting from 1-19 and the other from 1-31, so that their combination would lead to patterns as long as the lowest common multiple of these numbers (589). This contributed to the long time scale structure of the output. However, the entire mechanism was dependent on the exact numbers used throughout the patch (e.g. the lengths of the counters’ cycles) and it could only have been arrived at through endless intuitive exploration. There were no elegant algorithms or specific procedures that I could absorb into my practise, as a jazz player might learn Charlie Parker lines.

My gut feeling, though, was that there must exist general methods that a musician could use to design musical decision making on the arrangement-level time scale, at least for much electronic and improvised music. Indeed, I was absolutely sure that my own limited musical sophistication could be readily captured in an agent and that a tool that would allow me to do so would provide the missing ingredient for my creation of generative Max patch compositions. Thus, I started investigating, and first tried the approaches described in Chapters 3 and 4. These studies were formative. In particular, they led to the idea of programming by example—that a system could learn from examples. However, it took some time to realise that because of the sheer breadth of possible Max patches one might construct, to avoid having to supply endless examples, the musician would have to give some extra information to the system about what it could and could not do, about which elements of the examples were important and which could be ignored. This insight led to the Agent Designer Toolkit (see Chapter 5 onwards), the software that I wrote over the latter
two thirds of this thesis work. I don’t know if I could use it to make a musical agent to play Licfflii, but I have found it extremely rewarding to use, and when it is made available, I hope others will too.
Chapter 2

Background

2.1 Ableton Live and Max ................................. 46
  2.1.1 Ableton Live .................................................. 47
  2.1.2 Max ................................................................. 49

2.2 Machine Learning ............................................. 50
  2.2.1 The Machine Learning Workflow ......................... 51
  2.2.2 Interactive Machine Learning ............................... 52

2.3 Fixed and Variable Order Markov Models .................. 53
  2.3.1 Markov Models .................................................... 54
  2.3.2 Variable-Order Markov Models .............................. 57
  2.3.3 The Continuator .................................................. 61

2.4 Conclusion ..................................................... 63
In this chapter, we introduce the principal concepts and terminology related to a number of topics that are relevant to much of the remainder of this thesis. We begin with an overview of two software packages commonly used in contemporary electronic music: Ableton Live and Max (Section 2.1). We then give an overview of machine learning, by which models of real-world phenomena can be inferred from a data set (Section 2.2). Finally, we present the basic theory of Markov models, specifically, variable-order Markov models, which are among the models that can be arrived at using machine learning methods (Section 2.3). Included in this is a discussion of the Continuator, a particular application of variable-order Markov models to modelling a musician’s playing style, for reference in later chapters (Section 2.3.3).

2.1 Ableton Live and Max

In this section, we describe two platforms for electronic music, Ableton Live and Max. Both are used extensively in the systems reported in this thesis and therefore, it is helpful to establish the relevant concepts and terminology.

A more important reason for discussing Ableton Live, however, is that it provides further motivation for developing tools for designing musical agents. Ableton Live is music production and performance software and central to performing with it is the activity of sequencing and juxtaposing musical phrases and patterns, i.e. arrangement-level musical decision making. Since Ableton Live is readily extensible, it provides a software framework in which to develop generative and interactive music systems. In fact, interactive systems based on Ableton Live have already been developed (e.g. [38]).

To make the creative possibilities of generative music systems available to non-programming end users, all that is required is a suitable tool for creating musical agents (i.e. arrangement-level musical decision makers). That Ableton Live is an
extremely popular software package\(^1\) provides significant motivation for developing such a tool. Moreover, by providing a feature extractor, interactive systems could also be created. For example, a feature extractor might compute descriptors of an incoming audio or MIDI stream, or more straightforwardly, detect a musician’s activities within the software and allowing these to influence an agent’s decision making.

### 2.1.1 Ableton Live

Ableton Live is a music performance and production environment. While it can be thought of as a traditional digital audio workstation (DAW, see e.g. [113]), it has features that constitute important differences and make it a particularly suited for live electronic music performance. The most important of these is the **session view**, using which a musician can perform live music by sequencing pre-composed musical elements, such as MIDI and audio segments\(^2\).

The terminology associated with Ableton Live and particularly the session view is as follows (see Figure 2.1). The entire collection of pre-composed musical elements, as well as audio effects and software instruments, is referred to as a **Live set**. The pre-composed musical elements are called **clips** and these are arranged in **tracks** in which only one clip can play at a time; each track corresponds to a single audio signal chain. With reference to Figure 2.1, a clip can be **triggered** by clicking on the ‘play’ icon associated with it. Note that in the default configuration of Ableton Live, this does not start the clip playing immediately, but instead it starts at the beginning of the next musical bar. This makes it straightforward to maintain synchrony between clips. It also makes it possible to trigger multiple clips simultaneously, since they can be queued up over the course of a bar to begin at the start of the next one.

---

\(^1\)At the time of writing, the Digital Music Doctor website lists Ableton Live as the second most popular music software package [70]. In addition, a paper published in 2010 quotes an estimate on the publisher’s website of ‘several hundred thousand users’ [105], though this appears to have been removed from the website in the interim.

\(^2\)While Ableton Live is probably most widely used contemporary software package supporting this type of control, the idea is not without precedent. For example, similar features were available in MIDI software published by *Dr. T’s Music Software* in the 1980s [173].
Chapter 2. Background

Figure 2.1: The Ableton Live session view. The grid in which clips are placed takes up most of the space. Each coloured rectangle is a clip, and each column of clips corresponds to a track. There is no significance attached to the colour of a clip. Green triangles indicate which clips are playing. Note that there is no clip playing in the seventh or ninth tracks.

The simultaneous triggering of clips is also supported by the scenes feature of the user-interface: clicking on one of the play icons in the Master track (the rightmost column in the figure) starts all of the clips in the corresponding row playing. For example, in the Live set shown in the figure, clicking on the first scene (i.e. the first play icon) would start the first clip playing in each of the first three tracks and stop any clips playing in other tracks (since only the first three tracks have clips in the first position).

The features described above indicate the extent to which Ableton Live is designed for arrangement-level musical decision making. That electronic music is commonly performed in this way is also evidenced by the pervasiveness of hardware control devices such as the one shown in Figure 2.2. It is designed to reflect the Ableton Live session view, with a grid of buttons available to trigger clips in a performance context. A wide range of similar controllers are available.

Finally, as indicated above, the functionality of Ableton Live can be extended using suitable plug-ins. In particular, it is possible to use specially developed programs written in Max (see below) as plug-ins for Ableton Live. Such plug-ins are referred to as Max for Live Devices and they are developed using the Max for Live
2.1. Ableton Live and Max

Figure 2.2: The Ableton Live Push controller. Photo by Ralf Kleinermanns for Ableton; used with permission.

API³.

2.1.2 Max

Max is a graphical programming language for developing interactive and generative multimedia systems. In-depth accounts of the software can be found in [52, 120, 181]. Here we give a brief overview of the Max programming paradigm and the essential terminology.

A program in Max is referred to as a patch or Max patch⁴. A Max patch comprises a set of objects arranged in a two-dimensional space and connected using patch cords along which messages (primitive data types), audio signals or video signals may be passed. The computation performed by a patch is determined by the set of objects used; the values to which their internal parameters, if any, have been set; their spatial arrangement; and the configuration of the patch cords connecting them. There are objects to perform a wide range of functions from basic mathematical operations to sophisticated processing of audio or video signals. Essentially, each

³See: cycling74.com/docs/max6/dynamic/c74_docs.html#live_api
⁴The term patcher is also used.
object can be seen as software module with Max providing a framework for objects to communicate and share resources. A Max object may have its own graphical user interface (GUI) that offers additional ways by which a user can interact with it. Such a GUI can be graphically embedded in the Max patch itself, or it may appear in a separate window. Max includes a library of standard objects but its functionality can be extended by using third party objects or by creating new objects. Objects not included in the standard Max library are known as externals and they may be developed using C/C++, Java or Javascript.

We also note that it is common for electronic music practitioners to develop their own systems for live performance using Max. Furthermore, anecdotally, such systems frequently support arrangement-level musical decision making with live performance involving the juxtaposition and parametrisation of different sound-producing modules, each producing its own stream of audio. Thus, a tool for creating musical agents would be relevant to Max as well.

\subsection{2.2 Machine Learning}

In machine learning, an algorithm is used to infer the parameters of a model using a data set (see, e.g. [18] for a comprehensive treatment). During the learning process, the model is said to be trained and we will use the term training data set to refer to the data used during the training procedure. In supervised machine learning, the model is a function that maps inputs to desired outputs. A supervised machine learning algorithm requires a training data set comprising inputs and their corresponding outputs, and the parameters of the model are learnt such that it (ideally) maps new inputs (i.e. ones not found in the training data set) to the correct outputs. If the outputs of the model are categorical, the supervised machine learning algorithm is being applied to a classification problem, whereas if they are continuous, it is being applied to a regression problem. Unsupervised machine learning is different from its supervised counterpart in that the training data set is supplied without any correct
outputs and the parameters of a model are inferred so that the model describes the data in some way; this is also known as *pattern discovery*. Both supervised and unsupervised machine learning algorithms can be either *generative* or *discriminative*. In the former case, the (trained) model can be used to generate new data with characteristics similar to those in the training data set, whereas in the latter case it cannot.

In addition, a machine learning algorithm may be *instance-based*, meaning that it does not derive an abstract model from the training data set, but rather examines the data directly when required to make a decision. A simple example of this is a supervised *nearest-neighbour* algorithm applied to a classification problem. In this case the training data set comprises example points in some space, each with a classification label attached. To classify a new point, the algorithm simply finds the point in the training data set that is closest to the new point and outputs the label associated with it.

Finally, in machine learning, the term *overfitting* refers to training a model such that it describes patterns that are specific to the training data set, rather than the patterns that underly it. In contrast, *underfitting* refers to training a model such that it fails to capture the patterns underlying the training data set in sufficient detail. One of the perennial challenges in machine learning is to achieve a balance between overfitting and underfitting, that is, to capture the patterns underlying the training data set in good detail, without capturing the patterns specific to the training data set (for more details, see, e.g. [92, Ch. 7]).

### 2.2.1 The Machine Learning Workflow

The standard machine learning workflow is as follows (see Figure 2.3). The user typically begins with a training data set [stage 1]. He then configures the machine learning algorithms in two ways. First, he selects the features of the training data that will form the input [stage 2 (i)]. Features are properties of the training data set that result from a pre-processing stage. For example, in a computer vision problem
the training data might comprise greyscale digital images represented by matrices of pixel brightness values. It might be useful to extract the average brightness as a feature, or the percentage of pixels with a brightness greater than some threshold, for example. The second way in which the machine learning algorithms must be configured is to choose parameters for the algorithms themselves [stage 2 (ii)]. These parameters are entirely specific to the algorithms being used. The final stages of the standard machine learning workflow are to train a model according to the chosen configuration [stage 3], and then to evaluate it [stage 4] to decide if it needs further improvement [return to stage 2] or if it is ready to be used.

### 2.2.2 Interactive Machine Learning

The feature selection phase of the above workflow is of critical importance. Selecting unsuitable features can adversely affect the outcome to a great extent, and this can be an obstacle for users since effective choice of features requires expertise both in machine learning and in the problem domain. To circumvent this, Fails and Olsen proposed Interactive Machine Learning (IML) [78].

Interactive machine learning is applicable when the training data set is not immutable, but can be modified and augmented as necessary by a user (see Figure 2.4). The user begins by preparing an initial training data set [stage 1]. Then, during the training phase [stage 2], a large number of features are computed opaquely by the machine learning algorithms. In order that the algorithms can select the important
2.3 Fixed and Variable Order Markov Models

Since Lejaren Hiller’s early experiments with (fixed-order) Markov models [95], they have been widely used in computer music and algorithmic composition (see, e.g. [7] for a review). More recently, variable-order Markov models (VMMs) have been applied by Pachet, to learn individual’s instrumental playing styles [136] (see below). From Chapter 4 onwards, concepts relating to Markov models, and particularly VMMs, will be frequently referred to, and so we give an overview of the related theory here\(^5\). In addition, we outline Pachet’s Continuator software (cited above), also for reference in later chapters.

\(^5\) Another model in the Markov family, the partially observable Markov decision process is discussed in Chapter 3, however it is only relevant to that chapter and so it is introduced there.
2.3.1 Markov Models

Markov models are examples of history-based methods, 'wherein any features computable from preceding events can be used to condition the probability of the current event' [61]. A Markov model describes a system which may exist at any time in one of a set, $S$, comprising $N$ discrete states, $S_1, S_2, ..., S_N$. At regular instants in time, $t = 1, 2, ...$, the system transitions from one state to the next in probabilistic manner. The state at time $t$ is denoted $q_t$, thus, at time $t$, the system transitions from the previous state, $q_{t-1}$, to a new state, $q_t$. In general, the probability distribution from which $q_t$ is drawn depends on all previous states of the system, $q_1, ..., q_{t-1}$, but if the Markov property holds, the distribution only depends on the previous state, $q_{t-1}$. That is,

$$P(q_t = S_j | q_{t-1} = S_i ..., q_1 = S_k) = P(q_t = S_j | q_{t-1} = S_i), \quad 1 \leq i, j \leq N,$$  (2.1)

where the notation $P(A|B)$ is used to represent the probability that an event $A$ will occur, given that a condition $B$ is true. It is assumed that the probabilities do not change over time, and so a Markov model can be entirely described by a state transition matrix, $\mathcal{T}$ (the symbol is a calligraphic ‘T’) and an initial probability vector, $s_0$.

The elements of the state transition matrix are given by

$$\mathcal{T}(i, j) = P(q_t = S_j | q_{t-1} = S_i), \quad 1 \leq i, j \leq N,$$  (2.2)

and those of the initial probability vector are given by

$$s_0(i) = P(q_1 = S_i).$$  (2.3)

The latter represents a discrete probability distribution from which the first state an a sequence is drawn.
2.3. Fixed and Variable Order Markov Models

An Example of Melody Generation Using a Markov Model

It is straightforward to use a Markov model of this sort to generate musical melodies. The states might correspond to musical pitches. To generate a melody, an initial pitch is drawn from $s_0$, and after that successive pitches are drawn from $\mathcal{T}$. More complex knowledge representations (i.e., representations of musical information) are also possible, such as to use tuples of pitch and duration as the Markov states.

The model used to generate a melody may be created by hand, or it may be derived from analysis of an existing melody, or group of melodies. In the latter case, the Markov model is being used in the context of machine learning (see Section 2.2 above), and it is trained using a training data set comprising representations of existing melodies. For example, the melody shown in Figure 2.5 could be used as a training data set (albeit a very small one). A suitable Markov model might comprise five states, one for each note of the A-minor pentatonic scale used. The initial probability vector would then contain all zeros, except $P(q_1 = A) = 1$, i.e., the first note of the generated melody must be A, since no alternatives are found in the training data set. The state transition matrix would be calculated by first entering for each element, $\mathcal{T}(i, j)$, a count of the number of times the note corresponding to state $S_i$ is followed by that corresponding to the state $S_j$, and then normalizing each row to unity. The matrix that would result from this procedure is given in Table 2.1.

![Figure 2.5: A melody in the A-minor pentatonic scale.](image)

The model presented so far is known as a first-order Markov model. In the context of music generation, one important characteristic of this model is that the probability distribution describing the duration for which the model will remain outputting the same state is exponential in shape. This can be shown by observing

---

6In linguistics, first-order Markov models are also known as finite-state grammars.
Table 2.1: The state transition matrix derived from the melody in Figure 2.5. Each row corresponds to a state at time \((t - 1)\) and each column, to a state at time \(t\). For example if the previous state was \(C\), there is a a probability of \(\frac{1}{3}\) that it will be succeeded by an \(A\), and a probability of \(\frac{2}{3}\) that it will be succeeded by a \(D\).

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1/3</td>
<td>0</td>
<td>2/3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1/3</td>
<td>0</td>
<td>1/3</td>
<td>1/3</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1/3</td>
<td>1/3</td>
<td>0</td>
<td>1/3</td>
</tr>
</tbody>
</table>

the fact that once the system enters a particular state, \(S_i\) at time \(t\), the probability that it will remain in that state for \(n\) more time steps is given by:

\[
P(q_{t+1} = S_i, \ldots, q_{t+n} = S_i, q_{t+n+1} \neq S_i) = \Xi(i, i)^n(1 - \Xi(i, i)).
\]  

(2.4)

Furthermore, the expected value for \(n\) is given by

\[
E\{n\} = \sum_{n=0}^{\infty} nP(n) = \sum_{n=0}^{\infty} n\Xi(i, i)^n(1 - \Xi(i, i)).
\]  

(2.5)

A closed-form expression for \(E\{n\}\) can be obtained by taking the derivative of the infinite geometric series, as follows:

\[
\sum_{n=0}^{\infty} x^n = \frac{1}{1 - x}
\]

\[
\Rightarrow \sum_{n=0}^{\infty} nx^{n-1} = \frac{1}{(1 - x)^2}
\]

\[
\Rightarrow \sum_{n=0}^{\infty} nx^n = \frac{x}{(1 - x)^2}.
\]

Now, substituting \(\Xi(i, i)\) for \(x\), and multiplying by \((1 - \Xi(i, i))\) gives:

\[
E\{n\} = \frac{\Xi(i, i)(1 - \Xi(i, i))}{(1 - \Xi(i, i))^2} = \frac{\Xi(i, i)}{1 - \Xi(i, i)}.
\]  

(2.6)

In some musical contexts, this will clearly be unsuitable. For example, as noted
in [176], in situations where durations are related to metric structure and other music features not captured by the model. One way to mitigate this problem is to use higher-order models.

**Higher-Order Markov Models**

Higher order Markov models are those in which the state transitions depend, not just on the previous state, but on a longer history of states. In general, in an \( n \)th-order Markov model:

\[
P(q_t = S_j | q_{t-1} = S_i, ..., q_1 = S_k) = P(q_t = S_j | q_{t-1} = S_i, ..., q_{t-n} = S_l).
\]  

Higher-order models can capture more complex patterns. When used in a machine learning context, higher-order models tend to produce output that is more similar to the training data set\(^8\). This increased modelling capability requires more computational resources; more storage is required for the transition probability matrix since the model requires a probability distribution for each combination of \( n \) previous states. The storage required for an \( n \)th-order Markov model is proportional to \( N^{n+1} \).

In addition to the storage required, if a higher-order Markov model is to be derived from existing data—melodies, for example—then more data will be required. This is because examples of every combination of \( n \) notes must be found in the melodies. If insufficient data is provided, many of the entries in the transition probability matrix will be zero.

**2.3.2 Variable-Order Markov Models**

Variable order Markov models are essentially an extremely efficient way of implementing higher-order Markov models [149]. A tree structure is used to store the training data set in such a way that it is computationally inexpensive to search for a

---

\(^7\)Higher-order Markov models are also referred to as \( n \)-grams. An \( n \)-gram is equivalent to an \((n - 1)\)th order Markov model.

\(^8\)Some of Lejaren Hiller’s experiments involved juxtaposing material generated by 0th and 1st order models [94].
given history of events, in order to retrieve the probability distribution from which
the next event may be sampled. Furthermore, no memory is wasted storing the
zero-probabilities associated with event sequences not found in the training data set.

A VMM tree is shown in Figure 2.6. This was constructed from the A-minor
melody in Figure 2.5 according to the procedure described by Pachet in [136]. This
procedure will not be recounted here, however, we do describe how the tree, once
constructed, can be used to generate continuations for a given sequence of notes.
First, beginning at the root, R, the tree is descended as far as possible according to
the sequence of notes in reverse order (i.e. beginning with the one most recently
played). For example, if the sequence of notes was GAC, then beginning at the root,
a step is taken to the node C (marked with a green oval), and then from there to
the node A (marked with a red oval). However there is no node G connected to the
node A, so no more steps can be taken. At this point, the possible continuations of
the sequence AC are read from the list in braces attached to the node A to which the
final step was taken. This list contains the notes A and D and one of these is chosen
randomly. The list may contain multiple instances of some notes, and this would
alter the probability of different continuations being chosen. The node C, marked
with a green oval in the figure, is an example of this.

The above example illustrates how VMMs vary the order of the model according
to the data available. Because the sequence GAC had not been observed in the
training data, but AC had, the model was limited to second order. However, a 5th-
order continuation can be found for the sequence EGGEG, since this was observed
in the training data (the continuation must be the note C, as indicated by the node
marked with a purple oval in the figure). In general, a VMM uses the highest order
possible given the history and the training data set, but to increase the variety of
possible output—to prevent the VMM from simply copying the training data set—a
limit can be set on the maximum order used. In the remainder of this thesis, when a
limit, $n$, is placed on the order used by a VMM, we shall refer to it as an $n$th-order
VMM. The minimum order is 0, and a value is drawn from a 0th-order VMM by
Figure 2.6: A VMM tree that has been constructed from the melody in Figure 2.5.
randomly choosing one of the continuations from the list attached to the root node (R in Figure 2.6). We conclude this discussion of VMMs by noting two subtle issues related to their use.

**The History-Length Limits the VMM Order**

A subtle issue arises from the fact that the maximum order at which an nth-order VMM can operate is limited not only by n, but also by the history length, i.e. the number of values that have already been generated. Consider the following training sequence:

\[
\{1, 2, 3, 4, 0, 1, 2, 7, 2, 3, 9, 0, \ldots \}
\]

Now, suppose that it is required that, where the variable in question is concerned, generated sequences must begin with \{1, 2, 3, 4\}. Using a 3rd-order (or higher) VMM to model the variable is not sufficient to guarantee this. While the sequence must begin with a 1, which must be followed by a 2 (no other symbol ever follows a 1), the 2 may be followed by either a 3 or a 7. This is because, after the sequence \{1, 2\} has been generated, the model is limited to 2nd-order (the history length).

An effective solution to this problem is to introduce an ‘invisible’ starting state to the training sequence for each VMM. Thus, the training sequence above becomes

\[
\{S, 1, 2, 3, 4, 0, 1, 2, 7, 2, 3, 9, 0, \ldots \},
\]

where S is the start state. Now the generated sequence must begin with \{S, 1, 2\}, for which the only continuation is 3.

**A VMM can get ‘Stuck’ on a Particular Value**

Certain training data sets can cause generated sequences to get ‘stuck’ on a particular value. Consider the following sequence used to train an 3rd-order VMM:

\[
\{0, 1, 2, 2, 0, 1, 2, 2, 2\}.
\]
The sequence of values generated by this VMM can become an endless repetition of the value 2. This is because in the training data set, the only continuation of the length-3 sequence \{2, 2, 2\} is 2. In general, this problem arises when there is an integer, \(k\), such that there are at least \((k + 1)\) repetitions of a value at the end of a sequence and there are no other portions of the training data set containing more than \((k - 1)\) repetitions of the same value, and in addition, the VMM being used is of order \(k\) or greater\(^9\).

The simplest way to fix this problem is to lower the order of the VMM. However, in some contexts this may be undesirable. To mitigate the problem in such contexts, the VMM can be trained using ‘looped’ sequences of values. For example, instead of using the sequence above to train a 3rd-order VMM, the sequence

\[
\{0, 1, 2, 2, 0, 1, 2, 2, 2, 0, 1, 2, 2, 0, 1, 2, 2, 2, 2\}
\]

(created by concatenating two copies of the original sequence) would be used. There are now two possible continuations for the sequence \{2, 2, 2\}: 2 and an ‘escape value’, 0. This solution becomes less effective if there are far more than \(k\) repetitions of the value at the end of the sequence in the training data set, since the probability of the escape value being drawn is much lower.

### 2.3.3 The Continuator

The Continuator \([136, 137]\) is arguably the most well-known application of VMMs in computer music, and we provide an overview here for reference in later chapters. The Continuator was proposed as a system to ‘bridge the gap between interactive musical systems, limited in their ability to generate stylistically consistent material, and . . . music imitation systems, which are fundamentally not interactive’. It has two main modes of operation, a call and response mode in which the musician’s phrases

\(^9\)There are similar situations in which the VMM-generated sequence can get stuck in a repeating pattern, rather than on a single value. For example, a 3rd-order VMM trained on the sequence \{3, 1, 2, 3, 1, 2, 1, 2\} can get stuck on repeating the pattern \{1, 2, 1, 2, \ldots\}.\)
are ‘continued’ in their own style; and a collaborative mode in which a musician and the Continuator play simultaneously with the latter being influenced by the musician’s performance.

In the call and response mode, the Continuator works as follows. The incoming stream of notes is divided into phrases by detecting silences longer than a threshold duration. It is an adaptive threshold, being constantly modified according to the rate at which notes are being played. MIDI note sequences and global descriptors are extracted from each detected phrase. The MIDI note sequence corresponding to the incoming phrases are used to progressively build a set of VMMs. Each VMM in the set corresponds to a particular ‘reduction function’ which is a musically-informed approximate representation of the note data (e.g. just the sequence of pitches, ignoring duration and velocity). When the musician stops playing, the generator uses the musician’s note sequence as a seed (i.e. a history) from which to generate a new sequence using the learnt VMMs.

For generating new sequences, the VMMs are arranged in order of increasing level of approximation of their associated reduction functions. For example the first VMM (least approximation) might model the sequence of a multi-dimensional variable that is a tuple of the pitch, duration and velocity of the musical notes; the second VMM might model a variable comprised of just pitch and velocity; and the third might model a variable representing just the pitch. These VMMs are queried in order so that when a previously unseen musical situation arises and the first VMM cannot supply a continuation (i.e. the maximum available order is 0), the system uses a VMM associated with a reduction function with a greater level of approximation. Rather than outputting reduced representations directly from a VMM, the VMM-generated sequence indexes into the corpus of recorded MIDI data, so that the notes output have the subtle variation of the musician’s input (e.g. if the note A♭ is chosen, a randomly-chosen MIDI recording of the musician playing A♭ is output).

The collaborative mode differs from the call and response mode in that at each
2.4 Conclusion

In this chapter, we have provided an overview of two computer music software packages, Max and Ableton Live, that are relevant to much of the work in the remainder of this thesis. This was followed by a brief overview of some key concepts and terminology related to machine learning and interactive machine learning. In the context of machine learning applied to music performance, we discussed variable order Markov models (VMMs) and the Continuator, which uses VMMs to learn the note-level decision making behaviour of a human performer. This paradigm of using unsupervised machine learning to train a model of musical performance will be revisited frequently in the remainder of this thesis. In the next chapter, we discuss the first of three general purpose methods for designing arrangement-level musical decision making. It is based on another model in the Markov family: the partially observable Markov decision process.
Chapter 3

A Study of the Potential of the Partially Observable Markov Decision Process for Designing Musical Agents

3.1 Background ......................................................... 68
  3.1.1 Characteristics of the POMDP Model ............................. 68
  3.1.2 Related Work in Computer Music .............................. 68

3.2 Zero-Discount POMDPs for Designing Agent Responses .......... 69
  3.2.1 The Tonal Improvising System .................................. 70
  3.2.2 Using a POMDP to Choose the Key ............................. 72
  3.2.3 Software Implementation ........................................ 77
  3.2.4 How the POMDP Parameters Affect Behaviour ................. 78
  3.2.5 The Affordances of Zero-Discount POMDPs .................... 80

3.3 POMDPs for Designing Complex Musical Decision Making .......... 81
  3.3.1 The Potential for Designing Complex Behaviour .................. 81
  3.3.2 Designing the Reward Function ................................ 82

3.4 Conclusion ......................................................... 83
In this chapter, we address research question I (see Section 1.6.2), describing the first of three investigations into general purpose methods for the design of musical agents. We study the partially observable Markov decision process (POMDP), which is a mathematical model of the interaction between an agent and its environment, and its characteristics correspond well to our requirements (see Section 1.6.1). For example, to use the model requires no expertise in conventional computer programming. The (musical) goals of an agent can be set parametrically: it is required only to choose the numerical rewards associated with taking particular actions in particular circumstances. Once these rewards have been chosen, there exist structured methods for implementing a decision making behaviour in accordance with them. In addition, the POMDP has been used in a wide variety of application domains involving a commensurate variety of knowledge representations; it is general purpose in our terminology. Furthermore, it is suitable for real-time decision making. Somewhat less clear is the model’s potential for the design of generative decision making, that is, decision making in non-interactive scenarios. However, a closely related model, the (completely observable) Markov decision process (MDP) has been used in a generator of four-part harmony [184], for example. Thus, we proposed to investigate the use of POMDPs for interactive contexts and then, should it prove promising, to research its application to generative scenarios, perhaps using the work in [184] as a starting point.

The POMDP model arose in operations research as a framework for decision making under uncertainty. A POMDP-based agent can choose optimal actions with respect to a pre-defined set of goals, even without full knowledge of its environment. Amongst the decision making problems to which such agents have been applied, are robot navigation (choosing a path through a terrain) [161]; machine maintenance (optimising the inspection and servicing of equipment); marketing (using marketing resources efficiently); and automated search and rescue (finding moving targets
efficiently; see, e.g. [47]). To our knowledge, this is the first study of their use in musical interaction. As such, our aim was to investigate what design possibilities the POMDP model would afford a non-programming end user.

We address this in two parts, treating two different types of POMDP model separately. The first is the zero-discount POMDP model which can be used to design the responses of a system to changes in its environment. In this part, we describe a simple player-paradigm system that we developed. It uses a zero-discount POMDP in its decision maker. This serves as a proof-of-concept and also to illustrate certain design possibilities, such as the creation of musical agents that are ‘cautious’ or ‘risk-taking’ with regard to taking certain musical actions, as well as others with any of a wide variety of idiosyncratic preferences.

We then discuss the use of the more general non-zero-discount POMDP model. This model can support planning and anticipation and we consider its use for designing more complex musical behaviours. This is a theoretical discussion drawing from the POMDP and computer music literatures, in which we identify the potential advantages of using POMDP models and the related challenges. In particular, we argue that while the model is capable of supporting complex decision making, both the design of the reward function (the parameters that describe the musical goals of the system) and the specific choice of knowledge representation constitute significant obstacles. We conclude that in the context of the goals of this thesis, other methods need to be studied.

The structure of this chapter is as follows. We begin in Section 3.1 with a brief overview of the characteristics of the POMDP model, highlighting its apparent applicability to musical decision making. We also give a brief survey of the use of related models in computer music research. In Section 3.2, we describe a simple player-paradigm music system in which the musical agent—the arrangement-level musical decision maker—is based on a zero-discount POMDP. We describe the model and its software implementation, before illustrating how the parameters of the model can be conveniently used to design an agent’s responses. In Section 3.3,
we discuss the potential of general POMDP models to designing more complex behaviours, before concluding in Section 3.4.

3.1 Background

3.1.1 Characteristics of the POMDP Model

A thorough, mathematical description of the POMDP model is given in Section 3.2. Here, we give a brief overview of its characteristics. The POMDP model (zero-discount or not) supports decision making in a scenario in which the agent must take into account

1. the error rates of a sensor;

2. the behaviour of that with which the agent interacts (its environment) including its likely responses to the actions which the agent might take; and

3. a set of goals.

This corresponds well to arrangement-level musical decision making. For example, many of the player-paradigm systems referred to in Section 1.3 involve feature extractors that categorise the musician’s activities (e.g. [65, 130]). Such feature extractors are inevitably imperfect and a POMDP can take the related error rates into account. Less common are systems that include specific knowledge about the behaviour of the musician. However, the GRI system, for example, does include a first-order Markov model of the musician’s behaviour [130]. Finally, all systems feature—implicitly or not—the musical goals of their designers.

3.1.2 Related Work in Computer Music

Models related to the POMDP include hidden Markov models (HMMs) and Markov decision processes (MDPs). HMMs can also be used to take advantage of knowledge of musical behaviour. They have been used in this way to improve the identification
of chords and keys in recorded music [110]. However HMMs cannot account for the effects that an agent’s actions might have on a musician, nor can they be used to specify musical goals. MDPs can take these elements into account and as mentioned above, they have been used as the basis of a four-part harmony generator [184]. An initial investigation of their use in musical agents has been reported by Collins [58, 59], in the context of reinforcement learning (see, e.g. [165]) in which an agent adapts its behaviour during a performance according to a reinforcement signal, which is a real-time measure of its ‘success’. This work raised numerous questions relating to the selection of reinforcement signals and of appropriate knowledge representations. We discuss these issues in relation to this work in Section 3.3. Finally, much of the work in this chapter was previously described in [121] and subsequent to that publication, some preliminary work into the use of mixed observability Markov decision processes (MOMDPs, closely related to POMDPs) for modelling musical pitch contours was reported in [79].

### 3.2 Zero-Discount POMDPs for Designing Agent Responses

In this section, we study the POMDP model in an extremely simple and well-defined musical context to which it can be readily applied. More specifically, we study the zero-discount POMDP which, unlike the more general version of the model, does not support planning or anticipation of future events (details are given below). Instead, it can be used to design an agent’s choice of response to changes in an environment of which it has incomplete knowledge.

The decision making problem we consider is one in which a musical agent must choose the key in which a lower-level generator should play, based on imperfect information about the key in which a musician is playing, the latter arriving from a feature extractor that estimates the key from an incoming stream of MIDI notes. While this is quite dissimilar to the examples of arrangement-level musical decision
making highlighted in Chapter 2, it fits very neatly with the POMDP model and so it served as a reasonable starting point for our research.

In the following, we give an outline of a simple system involving a musical agent that chooses the musical key in which to play (Section 3.2.1). We then give a thorough account of the POMDP-based agent, including the mathematical details of the model itself (Section 3.2.2), and a brief overview of the software implementation (Section 3.2.3). We then illustrate the potential for designing agent responses (Section 3.2.4) and comment on the affordances of the zero-discount POMDP model in this regard (Section 3.2.5).

### 3.2.1 The Tonal Improvising System

We implemented a simple interactive music system in Max (see Section 2.1.2). We refer to it as the Tonal Improvising System (TIS, see Figure 3.1). It is designed for use by a musician playing a monophonic, MIDI-enabled instrument. He/she plays along with a metronome which sets the beat of the music. The TIS receives the MIDI note data associated with his performance.

Using the terminology introduced in Section 1.3, the feature extractor is a key-finding module and every two bars it estimates the key in which the musician has been playing over the previous two bars (this is on the time scale of arrangement-level musical decision making). The decision maker (i.e. musical agent) is a key-choosing module that uses this estimate to choose the key in which the TIS will play. It is in the key-choosing module that a POMDP is used. Finally, the generator is an improvisation-generating module that takes the chosen key as input, and outputs a synthesized ‘improvisation’ in that key. This system will now be described in detail.

The key-finding module is an implementation of the key-finding method due to Krumhansl [108]. Out of the large selection of algorithms available (see e.g. [170]), we chose this method for its simplicity and ease of implementation in Max. Our implementation works as follows. The MIDI note data produced by the musician’s performance is analysed in two-bar segments. For each two-bar segment, a 12-vector
is calculated, indicating the length of time spent during the segment in each pitch class. The correlation between this vector and each of 24 key-profiles (one for each major and minor key; for details, see [108]) is calculated. The key estimate for the segment is the key-profile which gives rise to the highest correlation.

The key-choosing module chooses the key in which the TIS will play for the following two bars. The simplest way to do this would be to use the key estimate directly. However instead a POMDP model is used to define a more sophisticated mapping from key estimate to the key in which the TIS should play (see next section). Each time the key is re-estimated, the key-choosing module outputs one of 25 possible instructions to the improvisation-generating module. There is one instruction of the form ‘play in key X’ for each musical key and an additional instruction which indicates that no specific key should be played. We refer to this as the non-specific key instruction.

The improvisation-generating module takes instructions from the key-choosing module as input. It remains silent when it receives the non-specific key instruction and for the remaining instructions it plays the root note of the specified key once on each beat. This extremely simple generator is of little musical interest, but was sufficient to experiment with the TIS. The output of the improvisation-generating module can be heard by the musician via a loudspeaker.
3.2.2 Using a POMDP to Choose the Key

The POMDP is a discrete-time model of the interaction between an agent and its environment. The key-choosing module is an agent and the environment with which it interacts is the musician. It perceives the musician through the key-finding module and it acts upon the musician by controlling the key used by the improvisation-generating module. According to the POMDP model (see Figure 3.2), the environment exists at time $t_i$ in a particular state, $s_i$. The agent makes an observation, $o_i$, at time $t_i$ which provides some information about the current state of the environment. The agent uses the observation to choose an action, $a_i$, to take. This action has an effect on the environment so that later on, at time $t_{i+1}$, the environment has transitioned to a new state, $s_{i+1}$, which is probabilistically dependant on the action taken and the previous state. The agent then makes a new observation, $o_{i+1}$ and chooses a new action, $a_{i+1}$, and so on. This process is governed by the parameters of the POMDP model which are denoted by the tuple $(S, O, A, T, Z, R, \gamma, b_0)$, where the fourth, fifth and sixth symbols are calligraphic T, Z and R, respectively. These quantities are defined in the following.

![Figure 3.2: The POMDP model. Arrows indicate probabilistic influence.](image)

The parameter $S$ is a set containing the discrete states in which the environment can exist. In the TIS, the state of the environment is a representation of the key in which the musician is playing. There are 25 states, one for each musical key and an inactive state which indicates that the musician is not playing. The parameter $O$ is a set containing the discrete observations that the agent can make. In the TIS, the observations are the possible outputs of the key-finding module. There are also
25 of these, one for each key and a silent observation to indicate that no notes have been played over the last two bars. However, note that the key-finding module is imperfect: the observation may not correspond to the actual key in which the musician is playing. Finally, the parameter $A$ contains the actions available to the agent. In the TIS, the set of actions contains the possible instructions that can be sent to the improvisation-generating module. As described in Section 3.2.1, there are also 25 of these, one for each key as well as the non-specific key instruction.

The parameter $T$ is the transition probability function. This describes the probabilistic dependence of the state of the environment at time $t_{i+1}$ on the state at time $t_i$ and the action taken at time $t_i$. That is, $T(s_{i+1}, s_i, a_i) = P(s_{i+1}|s_i, a_i)$, where $P(x)$ denotes the probability of $x$. The transition probabilities can be set by hand, or by some algorithm which learns the behaviour of the environment. In the TIS, they were set by hand according to a stylized notion of the behaviour of a musician playing in a western tonal idiom. The probabilities were set such that:

- it is most likely that the musician will remain in the same key, somewhat likely that he/she will move to a harmonically related key and least likely that he/she will move to a key that is not harmonically related;
- all keys are equally likely after a period of inactivity of one segment (i.e. two bars) or longer; and
- there is an increased likelihood that the musician will move to, or remain in the key being played by the TIS.

For example, if both the musician and the TIS are playing in G-Major, then it is most likely that the musician will remain in G-Major, though somewhat likely that he will move to C-Major, D-Major or E-minor (see Figure 3.3).

The parameter $Z$ is the observation probability function. This describes the probabilistic dependance of the observation made at time $t_i$, on the state at time $t_i$ and the action taken at time $t_{i-1}$. That is, $Z(o_i, s_i, a_{i-1}) = P(o_i|s_i, a_{i-1})$. For the TIS, the observation probabilities were simplified so that $Z(o_i, s_i, a_{i-1}) = P(o_i|s_i)$, i.e. the
Chapter 3. A Study of the Potential of the POMDP for Designing Musical Agents

Figure 3.3: Shown are the values used for \( \log(P(s_{i+1}|s_i, a_i)) \), where \( a_i \) corresponds to G-Major. The first row and first column correspond to the inactive state. The remaining 24 rows and columns correspond to the twelve major keys (denoted \( M \)) from C to B, followed by the twelve minor keys (denoted \( m \)) from C to B.

observations were made independent of the previous action. This is appropriate since the key-finding module is not directly influenced by the actions chosen by the key-choosing module. The probability of making the silent observation when the musician is in the inactive state was set to unity. For the other observations, the values of \( P(o_i|s_i) \) were set by estimating the error rates of the key-finding module. This was done as follows. For each state \( s \), corresponding to a key \( k_s \), five thousand sequences of 16 equal-length notes were generated. Each sequence was generated by using the Krumhansl key-profile (see [108]) for \( k_s \) as a discrete probability distribution and sampling from it 16 times. The key-finding module was then used to estimate the key of each sequence, and the probability, \( P(o|s) \), of making observation \( o \), corresponding to key \( k_o \), when the sequence was generated using the key-profile corresponding to the key \( k_s \), was calculated using

\[
P(o|s) = \frac{N_{k_s,k_o}}{5000} \tag{3.1}
\]
where \( N_{k_0, k_s} \) was the number of times the key-finder gave \( k_0 \) as its estimate for a sequence generated using the profile corresponding to key \( k_s \). The observation probabilities obtained in this way for the keys C-major and C-minor are shown in Figure 3.4.

![Figure 3.4: The estimated probabilities of each key being observed when the actual key is (a) C-major and (b) C-minor. Observations 1-12 are the 12 major keys, starting with C, and 13-24 are the 12 minor keys, starting with C. The most likely errors are those in which a key is mis-identified as a harmonically related key, for example, Eb-Major for C-minor.](image)

The parameter \( R \) is the reward function. This is a real-valued function of the current state and the chosen action, \( R(s, a) \) and it describes the benefit of the agent performing action \( a \) when the environment is in state \( s \). The agent uses the reward function to choose the optimal action to take (see below). In the TIS, the reward function was parameterised by four numbers. A positive reward, \( r^+ \) was gained for playing in the same key as the musician; a negative reward, \( r^- \), was gained for playing in a key other than the musician’s key; a reward of 0 was gained for taking the non-specific key action when the musician was silent; and finally, a small negative reward of \( r^{\Delta^-} \) was associated with taking the non-specific key action when the musician was playing in a particular key.

Usually, an agent acts so as to maximise the rewards obtained for actions taken over some time in the future. In many situations the rewards gained for future actions are considered less valuable than those gained for immediate actions. To allow for this, an additional parameter is included in the POMDP model. It is called the discount, \( \gamma \) (\( 0 \leq \gamma \leq 1 \)), and it describes the value of future rewards relative...
to immediate ones. A value of 1 means that future rewards are considered to be equal in value to immediate ones. A value of 0 means that only the immediate rewards have value. Non-zero discount models are used in situations where the agent is required to take into account the future states of the environment (these are discussed in Section 3.3). However, the TIS is only required to play in the correct key at the current time, so a discount of zero is appropriate.

A POMDP-based agent operates by maintaining a belief state, $b(s)$, which is a discrete probability distribution over the states in which the environment might exist (when the agent’s sensor—a key detector, for example—is imperfect, it does not ‘know’ for certain what the state of the environment is). The parameter $b_0$ is the initial belief state. After each action is taken and a new observation is made, the belief state is updated, according to the following equation, where $b_i(s_j)$ is the belief at a particular time, $t_i$, that the state at that time, $s_j$, is some particular state, $q$ [103]:

$$
b_i(q) = \frac{P(o_i|a_i, b_{i-1})}{P(o_i|a_i, b_{i-1})} \cdot \sum_{s \in S} P(q|a_{i-1}, b_{i-1}, s) \cdot P(s|a_{i-1}, b_{i-1}) \cdot \frac{P(a_i|a_{i-1}, b_{i-1})}{P(a_i|a_{i-1}, b_{i-1})}
$$

This calculation is straightforward, only requiring the observation and transition probability functions, ($\mathcal{O}$ and $\mathcal{T}$, respectively), along with records of the previous belief state, $b_{i-1}(s)$ and the previous action taken, $a_{i-1}$.

Once an observation has been made, and the belief state has been updated, a new action can be selected. To choose an action, the agent refers to a policy which is a mapping from belief state to action. The optimal policy is one that will maximise the rewards obtained. In general, finding the optimal policy is a computationally demanding task, albeit one which can be done offline (see, e.g. [161]). However, for models with $\gamma = 0$, such as that used in the TIS, the optimal action, $a^*$, is simply the...
one which maximises the expected immediate reward:

\[ a^* = \arg\max_{a \in A} \sum_{s \in S} b_i(s) \cdot r(s, a). \]  

(3.3)

### 3.2.3 Software Implementation

To implement the key-choosing module, we developed a new object for Max in C++. It is called `pomdp.decider` and it comprises functionality to

- store a POMDP model;
- maintain a belief state using Equation 3.2 above;
- calculate optimal actions using Equation 3.3 above in cases where the model has a discount of zero; and
- apply a user-supplied policy in cases where the model has a non-zero discount.

This functionality was tested using simple POMDP models and policies from the literature, such as those presented in [48, 161].

The `pomdp.decider` is used as follows. First, a POMDP model must be created and stored in a text file using the file format developed by Cassandra\(^1\). This file is then loaded into an instance of the `pomdp.decider` object in Max. If the model has a non-zero discount, a policy must be supplied by the user. There are a number of freely available POMDP solvers that can calculate optimal policies, including `zmdp` [161]. The `pomdp.decider` object has the ability to read policy files in the format output by `zmdp`. Once the model and policy (if required) have been loaded, the object is ready for use.

When in use, the `pomdp.decider` takes integers representing observations as input. Each time an observation is received, the object outputs an integer representing the optimal action, and an array of floating-point numbers to indicate the newly-updated belief state.

\(^1\)See: cassandra.org/pomdp/code/pomdp-file-spec.shtml.
Chapter 3. A Study of the Potential of the POMDP for Designing Musical Agents

<table>
<thead>
<tr>
<th>Agent #</th>
<th>$r^+$</th>
<th>$r^-$</th>
<th>$r^{\Delta}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>-100</td>
<td>-5</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>-10</td>
<td>-5</td>
</tr>
</tbody>
</table>

Table 3.1: The reward function parameters for two POMDP agents used in the TIS.

3.2.4 How the POMDP Parameters Affect Behaviour

The TIS is a toy system developed to experiment with the use of a POMDP as the basis of the decision making functionality of a musical agent. The decision making behaviour is affected by the transition probability function, the observation probability function and the reward function. In this section, we illustrate how these model parameters, respectively, can be used to design agent behaviour.

The main feature of the transition probability function described above is the high probability for the musician to remain in the same key. This introduces a damping effect on the change to the belief state when a new observation is made (the agent is less inclined to ‘believe’ a new observation indicating a change in key). This damping can be manipulated by varying $p_s$, which is the probability that the musician will remain in the same key. With $p_s$ set close to unity, the belief state changes very little, and as $p_s$ is reduced, the damping decreases. Similar damping effects are featured in the following.

To illustrate the effects of the observation probability function and the reward function, we compare the behaviour of two agents in two different scenarios (see Figure 3.5). The agents are identical but for their reward functions. The relative magnitude of $r^-$ (the negative reward for playing in a key other than that of the musician) to $r^+$ (the positive reward for playing in the same key as the musician) is much greater for the first agent (see Table 3.1). Each scenario begins with the belief state representing a probability close to unity for the state corresponding to C-Major and close to zero for all other states. In scenario 1, $o_1$ is the C-Major observation and all subsequent observations correspond to C#-Major. In scenario 2, $o_1$ is the C-Major observation and all subsequent observations correspond to A-minor.
### 3.2. Zero-Discount POMDPs for Designing Agent Responses

#### Figure 3.5: Shown are the belief states and actions of two agents in two separate scenarios. The belief states are shown as bar charts indicating the probability assigned to each state. The salient states are labelled. The symbols used for key observations and actions are self explanatory except for ‘---’ which denotes the non-specific key action (i.e. the TIS is silent). In scenario 1 (left half of the figure), the incoming observations change from C-Major to C#-Major. In scenario 2 (right half), they change from C-Major to A-minor. The belief state changes much more quickly in scenario 1, since mistaking C-Major for C#-Major is much less likely than mistaking C-Major for A-minor. The reward function of Agent 1 has a much greater relative magnitude of $r^-$ to $r^+$ than that of Agent 2. This causes Agent 1 to be much more ‘cautious’; it tends to take the non-specific key action unless it is ‘certain’ of the musician’s key.

<table>
<thead>
<tr>
<th>DP</th>
<th>Observation</th>
<th>Belief State</th>
<th>Agent 1</th>
<th>Agent 2</th>
<th>DP</th>
<th>Observation</th>
<th>Belief State</th>
<th>Agent 1</th>
<th>Agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>$o_i$</td>
<td>$b(\mathbf{s})$</td>
<td>$a_i$</td>
<td>$a'_i$</td>
<td>i</td>
<td>$o_i$</td>
<td>$b(\mathbf{s})$</td>
<td>$a_i$</td>
<td>$a'_i$</td>
</tr>
<tr>
<td>1</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>1</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>C-Maj</td>
</tr>
<tr>
<td>2</td>
<td>C#-Maj</td>
<td>---</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>2</td>
<td>A-min</td>
<td>A-min</td>
<td>C-Maj</td>
<td>C-Maj</td>
</tr>
<tr>
<td>3</td>
<td>C#-Maj</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>3</td>
<td>A-min</td>
<td>---</td>
<td>C-Maj</td>
<td>C-Maj</td>
</tr>
<tr>
<td>4</td>
<td>C#-Maj</td>
<td>---</td>
<td>C#-Maj</td>
<td>C-Maj</td>
<td>4</td>
<td>A-min</td>
<td>---</td>
<td>C-Maj</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C-Maj</td>
<td>5</td>
<td>A-min</td>
<td>---</td>
<td>C-Maj</td>
<td>---</td>
</tr>
<tr>
<td>6</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C-Maj</td>
<td>6</td>
<td>A-min</td>
<td>---</td>
<td>C-Maj</td>
<td>---</td>
</tr>
<tr>
<td>7</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C-Maj</td>
<td>7</td>
<td>A-min</td>
<td>---</td>
<td>A-min</td>
<td>---</td>
</tr>
<tr>
<td>8</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C-Maj</td>
<td>8</td>
<td>A-min</td>
<td>---</td>
<td>A-min</td>
<td>---</td>
</tr>
<tr>
<td>9</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C#-Maj</td>
<td>C-Maj</td>
<td>9</td>
<td>A-min</td>
<td>---</td>
<td>A-min</td>
<td>---</td>
</tr>
<tr>
<td>10</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>C-Maj</td>
<td>10</td>
<td>A-min</td>
<td>A-min</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Since the agents have identical transition probability functions and observation probability functions, their belief states are the same for the same sequence of observations. The observation probability function describes the low likelihood of the key-finding module mistaking C#-Major for C-Major (see Figure 3.4). Thus, in scenario 1 we see the belief state changing quite quickly as C#-Major observations are made. By contrast, in scenario 2 the belief state changes quite slowly, since the observation probability function describes a significant likelihood that C-Major will be mistaken for A-minor. We note that in the specific case of the TIS, the observation probability function and the transition probability function counteract each other to a certain extent. For example, the high likelihood described by the transition probability function of the musician moving from C-Major to F-Major is counteracted by the high likelihood described by the observation probability function of C-Major being mistaken for F-Major by the feature extractor.

Despite having identical belief states, the agents choose different actions because they have different reward functions. Agent 1 is designed to be quite ‘cautious’: the large magnitude of \( r^- \) means that it needs to be ‘certain’ of the musician’s key before choosing an action other than the non-specific key action; it remains silent, rather than risking to play in an incorrect key. This behaviour is unlike that of Agent 2, which is more ‘risk-taking’: the small magnitude of \( r^- \) means that the agent will choose the key with the highest probability even if that probability is not close to unity.

### 3.2.5 The Affordances of Zero-Discount POMDPs

This study, though far from exhaustive, demonstrates some potential uses of zero-discount POMDPs for designing the responses of a musical agent. Decision making under uncertainty can be optimised with respect to rewards set by the designer, and improved by prior knowledge of the musician’s behaviour and importantly, sensor accuracy. Moreover, this can be done in a structured way without requiring the designer to develop and implement ad-hoc heuristics that might otherwise be
3.3 POMDPs for Designing Complex Musical Decision Making

3.3.1 The Potential for Designing Complex Behaviour

POMDP models with non-zero discount values can result in much more complex interactive behaviours. This has been widely demonstrated in other domains, such as spoken dialogue management, where agents have been shown to plan and execute sophisticated strategies in order to achieve their goals (see [154], for example). The representation of music used in the TIS corresponds to well-established and straightforward musical expectations—the matching of musical keys—so it does not require elaborate planning. It is hard to imagine a non-zero discount scenario in which it would be desirable for the system to enter bad (low reward) key combinations in order to reach a good (high-reward) key combination at some time in the future.
future. However with a representation of music that allowed for a less prescriptive

treatment, a POMDP could certainly be used to design complex interactive beha-

viour. The advantages of this would be similar to those highlighted previously, the

foremost being that there are structured machine learning methods for finding the

optimal policy for selecting actions. In other words, instead of having to take a

heuristic approach to defining a control policy, the designer can focus on the concrete

aspects of the system which are the transition probability function, the observation

probability function and the reward function.

3.3.2 Designing the Reward Function

Despite this promise of the POMDP to mitigate the need for heuristics in designing
decision making behaviour, we envisage that for complex agents a difficult design
challenge would remain in defining the reward function. The traditional approach
is to set the rewards by hand, but this can be difficult or impractical in complex
scenarios, particularly those in which an agent is required to adhere to a particular
style [1]. There are a number of alternative approaches. One that is particularly suited
to musicians who are not experts in algorithm development, is inverse reinforcement
learning [133] (IRL). This is a family of machine learning techniques that can be used
for programming by example [118] (see next chapter), whereby behaviour is learnt from
a corpus of examples that are supplied by the designer. Using IRL, an appropriate
reward function is learnt from a set of examples. IRL has been successfully used for
learning driving style in a simple computer game [1] and in the real world [188], and
for learning styles of character movement in a virtual environment [111]. However,
algorithms for IRL are an active research area and standard, widely applicable tools
are not yet available.

In addition to IRL, we speculate that stochastic methods would also be effective.
For instance, a reward function could be randomly generated, or created using
evolutionary methods such as those used by Bown [31] to arrive at musically useful
artificial neural networks. This would allow for creative exploration of the space of
3.4 Conclusion

The POMDP framework has considerable potential for tuning the responses of player-paradigm systems. With reference to research question I, addressed in this chapter (see Section 1.6.2), the affordances of a zero-discount model to a non-programming end user include the design of systems that account for both the tendencies of the musician and the error rates of the sensor, as well as behavioural traits such as ‘cautious’ or ‘risk-taking’ as described above. In Chapter 10 we will speculate on the use of POMDPs at a macro-organisational level, for choosing between agents that have been designed by other means. However, for designing complex interactive behaviour it is less clear how to fulfil the apparent promise of the POMDP framework. Though models with non-zero discount values can give rise to such behaviour, open questions remain as to what knowledge representations will facilitate this and how to specify the reward function.

In particular, with regard to reward function specification, we envisage that hand-crafting would frequently be difficult or impractical. We have suggested two alternatives which themselves are contrasting and incompatible: stochastic methods and programming by example. The former is more suited to adherents of Perkis’ ‘wild system’ school, most interested in experimentation without a clear goal, while the latter is more suited to adherents of the ‘crafting’ school who have a particular idea in mind that they want to implement [36]. However, neither approach is proven.

For these reasons, and despite the advantages outlined above, we do not pursue the POMDP as a framework for designing arrangement-level musical decision making in this thesis. Instead, in the following chapter we explore further the paradigm of programming by example. Our aim is to use musician-supplied examples as a starting point for designing the behaviour of a musical agent.
Chapter 4

Programming Agent Behaviour by Example

4.1 Background ......................................................... 87
   4.1.1 PBE for Player-Paradigm Systems .......................... 88

4.2 A PBE System for Musical Decision Making ................. 90
   4.2.1 Overview of the Ensemble for Two Players .................. 91
   4.2.2 The Decision Making Procedure ............................ 94
   4.2.3 The Similarity Algorithm .................................. 96
   4.2.4 Software Implementation ................................. 97

4.3 Evaluation Methods ................................................. 99
   4.3.1 Real-time Interaction with a Musician ....................... 99
   4.3.2 Simulated Interaction .................................... 101

4.4 Results .......................................................... 102
   4.4.1 Observations Made During Interaction with a Musician .... 102
   4.4.2 Ability to Emulate the Style of Artificial Musicians .......... 103

4.5 Discussion ...................................................... 104

4.6 Conclusion ...................................................... 106
Programming by example (PBE) is a paradigm whereby a human trains a computer or robot to perform a particular task by supplying examples of doing the task himself or herself [118]. PBE systems are clearly relevant to our requirements since they have the potential to allow a user to specify the behaviour of a computational system without ever leaving the application domain (in this case, that of music performance). Thus, it might be possible to mitigate the need for any expertise, in conventional computer programming or otherwise, outside of the musical domain (Section 1.6.1). In modern PBE systems, it is common practise for machine learning techniques (see Section 2.2) to be used to infer a general model for decision making from the supplied examples; machine learning provides an alternative to heuristics or rule-based methods [132]. In the remainder of this thesis we use PBE to refer specifically to the paradigm in which systems use machine learning to infer general models from sets of examples.

In this chapter, we study the second method introduced in this thesis for end user design of arrangement-level musical decision making. We refer to the method as the similarity-based musical agent. It is a novel PBE system for arrangement-level musical decision making and it can be used as the decision maker of a player-paradigm interactive music system. The novelty of the system lies in the use of the similarity measure for sequential patterns in its matching procedure (see below), thus, this chapter addresses research question II (see Section 1.6.2).

The similarity-based musical agent uses an instance-based machine learning algorithm which is one that does not derive an abstract model from the training data set, but instead refers to it directly in order to make a decision [4]. An essential component of an instance-based machine learning algorithm is the matching procedure by which the system finds the segment of the training data set that is most similar to the current data. In many cases the matching procedure can be based on a straightforward metric. However, for interactive music applications more sophisticated
matching schemes are required. This is due to the complex knowledge representations used, and also to an assumption that there will in general be a paucity of example data available (more details are given below). Our PBE system differs from others in the computer music literature by using a matching procedure (based on the similarity measure for sequential patterns alluded to above) that is general purpose; it is not specific to a particular knowledge representation.

This chapter covers work previously presented in [122] and is structured as follows. In Section 4.1, we highlight some PBE systems in the interactive computer music literature, focusing particularly on player-paradigm systems and the matching procedures they employ. In Section 4.2, we describe a player-paradigm system based on our similarity-based musical agent, as well as the underlying decision making mechanism. We then describe the evaluation of the agent in the context of the player-paradigm system with respect to its style-emulation capabilities (Section 4.3). Specifically, we examine (i) how well the similarity-based musical agent can emulate demonstrated styles, and (ii) how the quality of this emulation is related to the number of examples provided (again, see research question II, Section 1.6.2). Our standpoint is that to warrant further study and development in the context of our requirements, a PBE system must be capable of capturing musical styles without overburdening the user by requiring a large number of examples. We present the results of this evaluation in Section 4.4 and in Section 4.5, we discuss the potential of the similarity-based musical agent in the context of the broader goals of this thesis.

4.1 Background

In the last chapter we referred to the use of inverse reinforcement learning as a method for PBE. However, it is more common to cast PBE either as a supervised or unsupervised machine learning problem (reinforcement learning is a distinct machine learning paradigm). These terms were introduced in Section 2.2.

PBE systems have been developed in a variety of interactive computer music
contexts. One application that has received considerable attention is the automatic creation of mappings from data representing physical gestures, to the parameters of sound synthesis or processing systems in the context of creating digital musical instruments. In Fiebrink’s Wekinator software, a supervised machine learning approach is taken [80]. To design an instrument, a user provides examples of inputs (e.g. mouse cursor positions) and their corresponding outputs (e.g. the parameters of an FM-synthesis algorithm [51]) and uses the machine learning algorithms exposed by the software to create generative mappings from one to the other. During a musical performance, the software computes the mapping in real time. The SARC EyesWeb Catalog [85] makes available a similar range of machine learning algorithms to users of the EyesWeb interactive platform [42] with an emphasis on multidimensional gestures that can occur over a variable time period.

4.1.1 PBE for Player-Paradigm Systems

In addition to these mapping applications, supervised machine learning algorithms can be used to train models for decision making. Typically a mapping is learnt between a state of the environment and an action [13]. While this approach could be taken for player-paradigm systems (the ‘state of the environment’ would correspond to a musical situation), it has not been prevalent. Instead, where supervised learning has been used in player-paradigm systems, it has been to train discriminative models to classify the behaviour of the musician in order to inform decision making carried out by other mechanisms (see, e.g. [65, 185]).

More frequently, for the development of player-paradigm systems we see unsupervised machine learning techniques being used to train generative models that are then used during performance to generate new musical data. This is the case with Omax [14, 63] and the Continuator [135, 136]. We have already seen that the Continuator (Section 2.3.3) trains a variable order Markov model using sequences of notes performed by a musician, and then uses the model to generate new melodies. In Omax, factor oracle algorithms (an alternative to VMMs for finding patterns
in a sequence of symbols; see [5, 112]) are used in a similar way. In related work, Chordia et al [50] trained VMMs on a data set comprising symbolic representations of North-Indian Tabla music; and they trained variable order hidden Markov models on a data set comprising audio representations of such music. However, in each case the aim was to arrive at a predictive model, rather than one for use in an interactive context.

These systems use detailed representations of sequences of musical notes. They take into account pitch, velocity and duration of notes (as well as other attributes depending on the use context). This reduces the likelihood of finding a perfect match in the training data set for a given sequence, because the variable used to represent each note has a large domain being formed by the Cartesian product of the domains of the various note attributes. In the research cited above, it is generally assumed that the training data set will be of such a size that for many sequences of notes, there will be no perfect match. Thus it is necessary to use an effective technique for finding the most similar segment in the training data set, in order to make good musical decisions when there is no perfect match.

In each of the above-cited examples of the use of VMMs and factor oracle algorithms, a variation of the multiple viewpoint scheme of Conklin and Witten [62] was used. This involves maintaining an ensemble of models that differ in the specificity of the knowledge representation they employ. To make a decision, the system begins by searching for a match using the most specific model. If a match cannot be found, it has recourse to a model using a more approximate knowledge representation, and then to one using a still more approximate knowledge representation, and so on as necessary (an example of this was given for the Continuator in section 2.3.3). The key point here is that the approximate knowledge representations are created using prior knowledge of human perception of musical note sequences; the system designers make judgements about the relative importance of various musical attributes. For example, pitch is generally taken to be a more important musical attribute than velocity, so in creating a knowledge representation that approximates a sequence
of musical notes, velocity would be omitted (or coarsely quantized) in deference to pitch.

The multiple viewpoint paradigm has been proven very effective in the *history-based* modelling methods (see Section 2.3) cited above. However, in a general purpose method for designing the behaviour of a musical agent, a pre-defined multiple viewpoint scheme cannot be used, because it is not known *a priori* what the input and output variables (\(p\) and \(q\) in Section 1.3) represent; it is not possible to arrive at reasoned approximations. The alternatives are (i) to allow the user to define the multiple viewpoint scheme as part of the design process, and (ii) to use a different matching procedure that does not require viewpoints to be defined. The *Agent Designer Toolkit*, which is software introduced in Chapter 5, supports a design activity that is related to the first of these. However, in the similarity-based musical agent, the latter approach was taken.

### 4.2 A PBE System for Musical Decision Making

In this section, we describe the similarity-based musical agent, which is a PBE system intended to perform arrangement-level musical decision making. It was created by taking the general approach used in the two successful player-paradigm systems cited above, Omax and the Continuator, and adapting it to our requirements. Both Omax and the Continuator can derive a model of a musician’s playing style from his performances and then use it to ‘improvise’ new music. However, as mentioned above, neither is general purpose, since both are designed specifically for generating and learning from symbolic data representing sequences of musical notes. Nonetheless, the fundamental idea of these systems is general purpose, being simply to store examples of a musician performing and to use those examples as a reference to make musical decisions. Specifically, to make a decision each system

1. creates a description of the current musical situation according to a particular knowledge representation;
4.2. A PBE System for Musical Decision Making

2. uses a matching procedure to find segments in the training data set similar to the current situation; and

3. chooses its next action so as to copy one of the most similar segments.

The similarity-based musical agent follows this procedure. Its novelty arises from the matching procedure used. Instead of using a multiple viewpoint system as a matching procedure in step 2 (above), we use an alternative similarity measure from the field of data-mining. It is referred to as the similarity measure for sequential patterns (abbreviated $S^2$MP) [157] and it is used to quantitatively describe the similarity between two sequences of sets of integers (i.e. sequences in which each element is a set of integers, or set sequences, for convenience). The data associated with a musical performance can take the form of a set sequence in which the sets correspond to snapshots of parameter values (i.e. the agent’s input and output variables: $p$ and $q$ in Section 1.3) taken at regular intervals. From a musical perspective, the $S^2$MP is an attractive measure because it takes into account both the contents of the sets (parameter values) and the order in which they occur (musical structure).

This section proceeds with an overview of a player-paradigm interactive music system with the similarity-based musical agent at its core (4.2.1). We then describe the decision making procedure (4.2.2) and the similarity algorithm on which it is based (4.2.3). Finally, we give details of the software implementation (4.2.4).

### 4.2.1 Overview of the Ensemble for Two Players

Here we describe a simple player-paradigm system with an arrangement-level decision making component based on finding matches in a training data set using the $S^2$MP measure. The system is called the Ensemble for two players and it was conceived as a framework for evaluating the similarity-based musical agent. We begin with an overview of the system.

The Ensemble for two players is a shared control system [71] whereby a musician shares control with a musical agent of an ensemble of virtual instruments. There are ten virtual instruments and the system supports arrangement-level control: the
musician and agent each manipulate a set of five binary-valued parameters (toggle switches) that turn on and off a corresponding set of five virtual instruments. The note sequences and rhythms played by the instruments are controlled at a lower level by pre-composed sequences and simple generative processes that follow a repeating, eight-bar harmonic structure and are synchronised to a central clock. (It is similar to a simple Ableton Live set in which the musician controls five tracks, and an agent controls another five; see Section 2.1.1.) Thus, the system is designed for real-time performance of a particular loop-based electronic music composition and the musician’s creative role evokes that of a musical arranger, deciding which instruments should play and when. One performance can differ from another in the order in which instruments are introduced and removed, in the combinations of instruments used together, and in the durations for which the combinations are left unchanged.

The system can be described in terms of the $PQfe$ model for player-paradigm interactive system introduced in Section 1.3 (see Figure 4.1). The input to the system, $Y$, which describes the musician’s performance, is the symbolic control data from the graphical user interface. The data are the states of the toggle switches for five virtual instruments and they are sent directly to the generator (via the extraction pathway) to control those instruments. They are also used directly to inform the decision maker, so the feature extractor simply passes them through unprocessed and $p$ is identical to $Y$. The decision maker uses the values to periodically update $q$, the values of the parameters under its control. Thus, the generator is parameterised by both $Y$, the control data for five of the instruments from the musician, and $q$, the control data for the remaining five instruments, from the decision maker. It also receives timing information from the central clock. Its output, $X$, is an audio signal that can be heard by the musician. We now outline the procedure used by the decision maker to choose parameter values, followed by a formal description of the algorithm.
4.2. A PBE System for Musical Decision Making

Figure 4.1: A schematic of the ensemble for two players. The blue text gives the terminology and notation introduced in Section 1.3 for the anatomy of an interactive system. The musician’s graphical control panel shows five toggle switches to turn on and off a set of five virtual instruments. The feature extractor in this system simply passes the parameter values to the decision maker, which uses them in order to choose its own actions. A central clock is used to (i) trigger the decision maker to update its parameter values once every eight-bars, (ii) synchronize the low-level processes controlling the virtual instruments, and (iii) update the graphical control panel.
4.2.2 The Decision Making Procedure

The \( S^2 \)MP technique is for measuring the similarity between two set sequences (as above: sequences of sets of integers). Musical control data can be stored as a set sequence by taking snapshots of the parameter values at regular instants during a performance: Each time a snapshot is taken, a corresponding set is created and the resulting set sequence represents the entire performance. (In the case of the Ensemble for two players, each set simply stores which virtual instruments were turned on at a particular time.) Thus, our expectation was that by using the \( S^2 \)MP, the similarity-based musical agent would make musical decisions consistent with the combinations of parameters found in the training data set (the sets) while taking into account musical structure on a longer time scale (the sequence of sets). This will be referred to again in Section 4.4. The decision making procedure is outlined here.

At regular decision points during the performance, the similarity-based musical agent searches the training data set to find the best match to the current musical situation and uses the match to choose new values for the parameters under its control. We use \( t_i \) (\( i = 1, 2, 3, \ldots \)) to denote the decision points. As mentioned above, the ensemble of virtual instruments is controlled by a set of switches. Each switch corresponds to a parameter that can take a value of unity (on) or zero (off). A set, \( M \), of these parameters is controlled by the musician and the remaining ones, \( C \), are controlled by the agent. Since the musician and agent each control five parameters, there are five elements each in \( M \) and \( C \). At time \( t_i \), the agent chooses parameter values according to (i) the training data set, \( \mathcal{D} \), comprising \( N \) example performances, \( \mathcal{D}^1, \ldots, \mathcal{D}^N \), and (ii) the parameter values used in the performance so far. The latter includes both musician- and agent-controlled parameters up to time \( t_{i-1} \) and the values of the musician-controlled parameters at time \( t_i \) just before the agent chooses new values for its parameters. Each example in the training data set is encoded as a set sequence. This is done by encoding the parameter values at each decision point as a set of integers indicating which switches were turned on at that time.

The new parameter settings are chosen as follows. At time \( t_i \), the agent forms a
4.2. A PBE System for Musical Decision Making

A performance set sequence, \( P \), from the parameter values at the last \( K \)-decision points (i.e. \( t_{i-K+1}, ..., t_i \); \( K \) is a positive integer) in the performance data. It does this by forming a set sequence in which the \( j \)th set \((j = 1, ..., K)\) encodes the set of switches that were turned on at \( t_{i-K+j} \). The agent then iterates over all length-\( K \) set sequences in the training data set and for each one, it calculates a similarity score using the \( S^2 \)MP algorithm (see Section 4.2.3 below). Since values for the agent-controlled parameters have not yet been chosen at \( t_i \), the final set in \( P \) is incomplete. Therefore, each time the similarity is calculated between a set sequence, \( Q \), from \( D \), and the performance set sequence, \( P \), the last set of \( P \) is modified to maximise the similarity with \( Q \), i.e. it is modified so that the agent-controlled parameter values encoded by its final set (that corresponding to \( t_i \)) are the same as those encoded by the last set of \( Q \). The best-matching set sequence, \( Q^* \), is that corresponding to the highest similarity score. Once \( Q^* \) has been found, the agent extracts the values of the parameters under its control, from the final set and uses these values at \( t_i \). A listing of the algorithm just described is given in the next sub-section.

The \( S^2 \)MP quantitatively describes the similarity between two set sequences. We identified it as a promising method for measuring the similarity between two sequences of multi-dimensional musical parameter data because it takes into account both the combinations of parameter values (i.e. the instantaneous juxtapositions of musical material) and the order in which they are used (i.e. the musical context). We give a brief outline here (see [157] for details). Given two set sequences a similarity score is calculated. This is a value between 0 and 1, where 1 indicates that the set sequences are identical, and 0 indicates that no set from the first set sequence has any elements in common with any set from the second set sequence. It is a weighted average of two measures of similarity: a mapping score and an order score. The mapping score describes the average similarity between individual sets in the two set sequences (i.e. parameter combinations), while the order score describes the extent to which the set sequences are ordered in the same way (the musical context).
4.2.3 The Similarity Algorithm

Here, we give a mathematical description of the method by which the similarity-based musical agent chooses new parameter values. First, we introduce some notation. The number of sets in a set sequence $S$ is denoted by $|S|$. The symbol $S_i$ is used to denote the $i$th set of $S$, and $S_{i:j}$ denotes the set sequence $\{S_i, S_{i+1}, \ldots, S_j\}$ (i.e. from the $i$th set to the $j$th set). Finally, we use $\{S, A\}$ to denote the set sequence resulting from the addition of a set $A$ to the end of a set sequence $S$.

The performance set sequence (see Section 4.2.2) is denoted $P$. Usually $P$ contains $K$-sets, but if there have not yet been $K$ decision points in the performance, then $P$ has as many sets as there have been decision points. The number of sets in $P$ is denoted $l$.

As discussed in the previous section, the final set of $P$ needs to be completed before the similarity between $P$ and a set sequence, $Q$, from the training data set, can be calculated. This is done by modifying the last set of $P$ so that it encodes the same agent-controlled parameter values as the last set in $Q$. We use $P'$ to denote this modified version of $P$. Formally,

$$P' = \{P_1, ..., (P_l \cup (C \cap Q_l))\},$$

where, again, $C$ is the set of parameters being controlled by the agent, and therefore $(C \cap Q_l)$ is the subset of the agent's instruments that are switched on in the final set of $Q$.

At a decision point, $t$, the agent must choose values for the parameters in $C$. This choice can be encoded by a set of integers, $C_1$, corresponding to the parameters that will be set to 1. To choose new values, the agent iterates over all length-$l$ set sequences in the training data set. For each of these, denoted $Q$, it creates a corresponding $P'$ and calculates the similarity score between $P'$ and $Q$. The parameter values chosen by the agent are those encoded in the final set in the set sequence which gives rise to the highest similarity score. The procedure just described is listed in Algorithm 1,
where the \( \text{sim} \) function (line 7) refers to the S\(^2\)MP algorithm.

**Algorithm 1** The similarity algorithm

```plaintext
1: \( s_{\text{max}} \leftarrow 0 \) \hspace{1em} \triangleright \text{Initialise the maximum similarity score, } s_{\text{max}}, \text{ to 0}
2: \( l \leftarrow |P| \) \hspace{1em} \triangleright \text{Set } l \text{ to the length of the performance set sequence}
3: \textbf{for } n = 1 \text{ to } N \textbf{ do} \hspace{1em} \triangleright \text{ } N \text{ is the number of examples in the training data set, } \mathcal{D}
4: \hspace{1em} \textbf{for } j = 1 \text{ to } (|\mathcal{D}^n| - l + 1) \textbf{ do} \hspace{1em} \triangleright \text{Iterate over length-} l \text{ sequences in the } n\text{th example}
5: \hspace{2em} \mathcal{Q} \leftarrow \mathcal{D}^n_{i+1} \hspace{1em} \triangleright \text{See text}
6: \hspace{2em} \mathcal{P}' \leftarrow \{P_i \cup (C \cap \mathcal{Q}_l + \mathcal{Q}_{l+1})\} \hspace{1em} \triangleright \text{See text}
7: \hspace{2em} s \leftarrow \text{sim}(\mathcal{P}', \mathcal{Q}) \hspace{1em} \triangleright \text{Calculate the similarity between } \mathcal{P}' \text{ and } \mathcal{Q}
8: \hspace{2em} \textbf{if } s > s_{\text{max}} \textbf{ then} \hspace{1em} \triangleright \text{If the similarity score is the highest so far}
9: \hspace{3em} s_{\text{max}} \leftarrow s \hspace{1em} \triangleright \text{Store it}
10: \hspace{2em} C_1 \leftarrow (C \cap \mathcal{Q}_l) \hspace{1em} \triangleright \text{Store the agent’s parameter values}
11: \hspace{2em} \textbf{end if}
12: \hspace{2em} \textbf{end for}
13: \hspace{2em} \textbf{end for}
```

### 4.2.4 Software Implementation

The *Ensemble for two players* shared control player-paradigm system was developed in Max, with the decision maker implemented in C++ as a new object called *am.interactor*. In order to enable the creation of the training data set, the system was developed with a ‘two-player mode’ so that the music can be performed by two musicians as well as by a musician and an agent. The interface for two musicians is shown in Figure 4.2. When the master clock is started, the system begins to repeat the preset musical form which is 8-bars in length. The low level processes that control the instruments follow this form throughout but as indicated above, the instruments only sound if they are switched on.
Figure 4.2: The three graphical user interface windows of the ensemble for two players, used in 'two-player mode'. (a) and (b) Two musicians control five virtual instruments each using either mouse or keyboard to toggle instruments on and off (the keys are shown for each instrument). A meter (in green) displays the sound output of each instrument (stereo instruments have a pair of meters). The harmonic output of the system follows an eight-bar form and the current position in the form is displayed to each musician. (c) The global control interface for starting and stopping the system, and recording examples.
4.3 Evaluation Methods

In this section we describe the methods used for evaluating the similarity-based musical agent as the decision maker of the Ensemble for two players just described. In the context of the broader goals of this thesis it was not required to exhaustively characterise the musical agent, but to assess its potential to form a foundation on which a general purpose tool for designing arrangement-level decision making behaviour might be developed. To this end, we focussed on the style-emulation capability of the similarity-based musical agent. Style emulation is the application to which PBE systems are more suited and as indicated in the chapter introduction, we view successful style emulation as a prerequisite for further study and development of the system. In particular, we were interested in the ability of the similarity-based musical agent to emulate realistic musical decision making and to do so without requiring an inordinate number of examples.

We evaluated the system in two ways. First we created a training data set by recording the control data associated with a number of performances using the system in two-player mode and then made observations while a single musician interacted with the similarity-based musical agent. Second, we investigated the extent to which the musical agent could learn the rules by which another simple rule-based agent performed. Further details are given below.

4.3.1 Real-time Interaction with a Musician

In the first evaluation, the agent interacted with a human musician in a shared control scenario. With respect to style emulation, our goals in this part were to investigate if the agent would act (i) to use combinations of instruments consistent with the training data set and (ii) to control the instruments in a manner consistent with the arrangement-level musical structure in the performance. To succeed in the latter, the agent would need to successfully differentiate, for example, between a buildup (i.e. a gradual increase in musical intensity [41, pp. 221-224]) and a breakdown (i.e. a
Figure 4.3: Six training examples used to train a similarity-based musical agent for real-time interaction with a human musician. The coloured lines indicate the periods during each performance for which the different instruments were switched on.

sudden reduction of intensity).

First, the system was used in two-player mode (see Section 4.2.4) to create a training data set comprising six example performances (see Figure 4.3). Then, a number of performances were conducted with the agent controlling five switches and a musician controlling the other five. Observations made during this part of the evaluation are given in the next section.
4.3. Evaluation Methods

4.3.2 Simulated Interaction

The second evaluation was done by creating ‘artificial musicians’ with which the similarity-based agent could interact. These were heuristics-based agents (H1 and H2) each of which operated according to a simple set of rules. Our aim here was to investigate if the similarity-based agent could use a training data set created by an artificial musician to act consistently with the underlying rules. Of particular interest was the dependency of the agent’s performance on the number of examples provided since as noted above, the number of examples required would be an important characteristic of a design tool.

The behavioural rules were in the form of dependencies between parameters, and chosen after inspection of the patterns visible in the training data set presented in the previous sub-section (Figure 4.3). The rules associated with H1 and H2 are indicated in Figure 4.4. In this figure, the nodes of the tree represent parameters and arrows indicate dependencies. Dotted arrows indicate inverse dependencies (e.g., in H1: 2 \rightarrow 1 causes 4 \rightarrow 0) and the numbers indicate probabilistic dependencies (i.e. the probability that a change will be induced is less than unity). Finally changes induced may occur after a delay of 0 or 1 or 2 time steps. The number of time steps in each case is chosen randomly.

For each trial in this quantitative evaluation, an artificial musician was used to generate a set of examples. Then an interaction was simulated between our similarity-based musical agent and the artificial musician. Performance was measured by calculating the percentage of times that a required parameter change was not present (as either a cause or effect), as a fraction of the total number of changes in the interaction. This percentage is denoted $p_b$.

The experimental conditions were chosen to investigate the effect on style emulation accuracy of (i) the number of training examples, $N$, (ii) $K$, the history-length to use when searching the training data set, and (iii) the choice of artificial musician, $H$. As a post-hoc analysis, we studied the effect of $C$, the choice of parameters under agent control. We used the conditions shown in Table 4.1 and for each one,
Figure 4.4: Tree structures showing the rules underlying artificial musicians (a) H1 and (b) H2. Arrows indicate that a change in the value of one parameter will induce a change in the value of another parameter within two time steps and with the probability shown beside each arrow. For example in H1, if parameter $3$ is set to 1, then there is a probability of 1 that parameter $8$ will be set to 1 within 2 time steps, and a probability of 0.6 that parameter $7$ will be set to 1 in the same time frame. Dotted arrows indicate ‘inverse’ dependencies, e.g. in H2 if parameter $6$ is set to 1, then parameter $9$ will be set to 0 within 2 time steps.

Table 4.1: The experimental conditions used for the second evaluation of the similarity-based musical agent in which it interacted with two ‘artificial musicians’ each of which operated according to a heuristic scheme.

<table>
<thead>
<tr>
<th>Condition #</th>
<th>K</th>
<th>C</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>${6,7,8,9,10}$</td>
<td>H1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>${2,3,4,5,6}$</td>
<td>H1</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>${6,7,8,9,10}$</td>
<td>H1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>${2,4,6,7,9}$</td>
<td>H2</td>
</tr>
</tbody>
</table>

we ran the experiment four times with different numbers of training examples, \(N = \{2,6,10,20\}\).

### 4.4 Results

#### 4.4.1 Observations Made During Interaction with a Musician

The observations made by the musician during real-time interaction with the similarity-based musical agent were as follows. First, the agent generally avoided combinations of instruments that were not found in the training data set. However, occasionally it
toggled an instrument on and off at successive decision points, when no such rapid changes had been demonstrated. We found that this occurred when the matching algorithm switched to different parts of the training data set that were similar except for the value of the parameter corresponding to the instrument in question. Second, the agent usually differentiated successfully between musical structures such as buildups and breakdowns. Thus, in general our expectations of using the similarity algorithm based on the $S^2$MP measure (appropriate parameter choices made with sensitivity to longer time scale musical structure) were fulfilled by these observations, while the rapid switching was unforeseen.

### 4.4.2 Ability to Emulate the Style of Artificial Musicians

The results of the interaction simulation experiments are summarised in Figure 4.5 with the experimental conditions shown in the legend. In general, accuracy increases ($p_b$ decreases) as the number of examples is raised. Two exceptions to this are conditions 2 and 3 where the mean $p_b$ value increases from $N = 10$ to $N = 20$. However the error bars are quite large for these conditions.

Conditions 1 and 2 differ only in the selection of parameters that the agent is controlling. We note that for every link to and from a parameter node in a heuristic tree, a parameter dependency must be satisfied. The total number of links to and from the nodes in Figure 4.4a associated with the parameters in condition 1, is 5 whereas that in Figure 4.4b (for condition 2) is 8. Therefore, condition 2 presents a more complex style-emulation task and this is reflected in higher $p_b$-values.

The large error bars indicate that performance varied considerably across trials. This is partly due to the fact that the training data set was randomly generated for each trial. In some cases the training data set did not provide a good representation of the variety of output that a given heuristic scheme could produce. This happened more frequently with lower values of $N$, and this explains why the error bars are larger for the $N = 2$ conditions.

In condition 1, $K = 4$, whereas in condition 2, $K = 7$. Performance is better
in condition 1. We suspect that this is because $K = 4$ is close to optimum for the heuristic schemes used, since parameter changes induce effects which always occur within two time steps.

For all of the experiments we qualitatively analysed the errors made by the similarity-based agent. We observed that when a rule was broken, it was usually because parameters were changed one or two time steps too early or too late, rather than because the wrong parameter was changed or no parameter was changed at all. In summary, the performance of the similarity-based musical agent as measured by the percentage of errors, $p_b$, improves as more examples are provided and is dependent both on the complexity of the style being learnt and the history-length used.

### 4.5 Discussion

The similarity-based musical agent provides a straightforward adaptation of the approach taken in Omax and the Continuator to a context in which the system embodies no assumptions about the knowledge representation being used. In the *Ensemble for two players* system used for our evaluation, the control parameters corresponded to switches controlling the activity of an ensemble of virtual instruments, but this was not a requirement. At the core of the similarity-based musical agent is
the S²MP. Apart from being a robust and computationally inexpensive algorithm, this measure was chosen because unlike other pattern similarity measures such as the edit distance [114], it takes into account both the contents of the pattern (i.e. the parameter values) and the order in which they occur. This means that immediate concerns (what combination of parameter values is appropriate here?) and structural concerns (is this part of a buildup, or a breakdown?) can be satisfied with a good match found using the S²MP.

While these benefits of the S²MP were apparent when the system interacted with a musician, there are a number of improvements that could be made. To prevent the occasional rapid switching on and off of instruments, heuristics could be included to ensure continuity in parameter values similar to that in the training data set. The limitation of the S²MP measure to binary-valued parameters could be circumvented using dummy variables by which multi-valued parameters are represented by sets of multiple binary-valued parameters (see, e.g. [92, p. 10]). In addition, the designer could be given more control over agent behaviour by introducing the facility for parameters to be weighted according to his/her conception of their musical importance, and for parameters to be grouped and modelled independently.

These additional features would begin to allow the similarity-based musical agent to support a design process in which the user could re-configure the parameters of the system to modify the agent’s behaviour. However, there are two important issues likely to be more difficult to resolve. First, the results in the previous section indicate that to learn even a simple style in which parameters have straightforward dependencies, more than twenty example performances are required to achieve an error rate of around 10% or less. While careful choice of examples (as opposed to randomly chosen ones as in our evaluation) might reduce this somewhat, many musical styles are likely to be significantly more complex, both in terms of the number of parameters and the ways in which they are manipulated. We envisage that a requirement to supply even a few tens of example performances is a significant barrier to the introduction of the similarity-based musical agent into a musician’s
creative workflow.

The second issue with respect to the requirements set out in Section 1.6.1 is that if used in a generative context, the similarity-based musical agent will simply act to duplicate one of the examples in the training data set. By limiting the history length, $K$, or by randomly selecting between segments of the training data set that are within a certain similarity ‘distance’ of the best match, it might be possible for the agent to produce novel performances by sequencing segments from different examples (this is essentially what happens in the Continuator and Omax, but on a shorter time scale), but the introduction of these features would not be without trade-offs and the agent would remain tightly constrained by the training data set.

4.6 Conclusion

We have described the similarity-based musical agent, a novel PBE system for use in designing and performing arrangement-level musical decision making. It provides a means by which a user can specify the behaviour of an arrangement-level musical decision maker without ever leaving the domain of musical performance. Moreover, it is computationally inexpensive and therefore suitable for use in parallel with other algorithmic processes. However, in the broader context of this thesis, the requirement for a large training data set constitutes a significant obstacle. With reference to research question II (see Section 1.6.2), this is the key result presented here.

As mentioned above, the similarity-based musical agent uses an instance-based learning algorithm, meaning that during the training phase it does not derive an abstract model from the training data set but instead stores the data to be queried directly. In some machine learning problems this is an advantage because the complexity of the model grows with the size of the training data set [156, p. 733]. However, the converse is also true and here it accounts for the first of the important issues identified above: the limited amount of training data restricts the complexity
of the model of musical style that can be modelled.

Using learning algorithms that are not instance-based, an abstract model is derived from the training data set. Such algorithms may appear to have little promise in our context, since complex models usually require large amounts of training data, while on the other hand, it seems reasonable to assume that no single simple model could be used for a wide variety of musical behaviours. However, in the following chapter, we present an alternative PBE approach using machine learning algorithms that are not instance-based. Relatively simple models are used, but they are defined and combined by the user for a particular musical context; we allow him to embed his knowledge into the model through custom configuration of the machine learning algorithms. In particular, we frame the feature selection phase of the machine learning workflow as part of the design process. This circumvents the limitations of models that are too simple to be generally useful, while still mitigating the need for large amounts of training data. It also allows for a variety of behaviours to be obtained from a single training data set.
Chapter 5

The Agent Designer Toolkit

5.1 Outline of the Research Related to the Agent Designer Toolkit . . 110
5.2 Outline of this Chapter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
5.3 Analysis of Performance Data from Three Music Systems . . . . 116
  5.3.1 The Montreal System ................................................................. 117
  5.3.2 An Electroacoustic Improvisation System .................................. 123
  5.3.3 A Drum and Bass Performance System ..................................... 126
  5.3.4 Analysis Outcomes ................................................................. 130
5.4 A Rationale for the Learning Components of the ADTK ............ 131
  5.4.1 Requirements of the Machine Learning Algorithms .................. 131
  5.4.2 Two Algorithms that Plausibly Meet the Requirements .............. 133
  5.4.3 Variable-Order Markov Models ................................................ 136
  5.4.4 Association Rule Learning ...................................................... 142
  5.4.5 Combining Markov Models and Rules to form a Model of Music Performance 147
  5.4.6 Extending the Model with Custom Variables ............................. 149
  5.4.7 How the Model Works in an Interactive Context ...................... 162
  5.4.8 The Learning Configuration .................................................. 163
5.5 Overview of the ADTK Software ................................................. 164
5.6 Discussion .................................................................................. 166
  5.6.1 Style Emulation ........................................................................ 167
  5.6.2 Discovery of New Styles .......................................................... 168
  5.6.3 Models of Music Performance .................................................. 170
  5.6.4 Interactive Machine Learning ................................................ 171
5.7 Conclusion .................................................................................. 172
This is the first of six chapters that deal exclusively with the Agent Designer Toolkit (ADTK), which is software for designing models of arrangement-level musical decision making (i.e. musical agents) for interactive and generative music systems. Collectively, these chapters deal with all five parts of research question III outlined in Section 1.6.2. Since these chapters are interrelated, we begin with an outline of all of the research concerning the ADTK that is reported in this thesis, ending with an indication of how it is distributed across this chapter and the five that follow. We then introduce the material presented in this chapter, specifically.

5.1 Outline of the Research Related to the Agent Designer Toolkit

The Agent Designer Toolkit is software for designing musical agents based on a set of example performances. Briefly, it can be used to create models of musical performance that comprise variable order Markov models (VMMs) for capturing patterns in the way variables change over time; association rules (i.e. \( A \implies B \)) for capturing dependencies between variables; and user-defined custom variables which are equivalent to features in machine learning and can be used to embed musical knowledge into an agent. The VMMs and association rules are learnt from a set of example performances by machine learning algorithms.

Variable order Markov models and association rules were initially selected as the core modelling elements for the ADTK out of usability considerations. Models related to each of these can be found in popular computer music software packages, and therefore, we envisaged that they could be fruitfully employed by a musician with no machine learning expertise. Moreover, we developed a procedure for using the two models in tandem and when used like this, they can capture two fundamental types of musical patterns: horizontal patterns, that describe the way
musical elements change over time (VMMs); and vertical patterns that describe the dependencies between elements that are used concurrently, such as which ones go well together, and which ones do not (association rules).

However, we found that even in combination, these two models were insufficient to capture certain important musical patterns. For example, it was not possible to capture buildups in intensity (as it was with the similarity-based musical agent presented in the previous chapter) because this requires context- or time-dependent rules. We therefore added functionality whereby a user could select features of the training data to include in the model in order to capture a variety of patterns including buildups. For example, the user could create a custom variable (i.e., a feature) representing the number of instruments playing. This variable could then be modelled using a VMM so that temporal patterns in the number of instruments playing could be learnt from the training data set. Thus, we envisaged, musical structures such as buildups and breakdowns could be captured, because these are typically characterised by the gradual introduction of instruments, and the sudden removal of instruments, respectively. In total, fifteen custom variable types (i.e., categories of features) were chosen to enable a variety of different musically relevant patterns to be captured. These were added to the ADTK such that they could be used individually or combined in various ways to create more complex features.

While association rules and custom variables can be used to model musical patterns, they are problematic because, unlike VMMs, they are not generative models: it is not straightforward to use them to generate new data. To find a set of variable values that is consistent with the relations imposed by a collection of association rules and custom variables, is to solve a constraint satisfaction problem, and such problems are not, in general, soluble subject to real-time constraints (i.e., in a strictly bounded length of time). However, we developed a method for generating new data using ADTK models (i.e., VMMs, association rules and custom variables) that can be used reliably in real time. It involves converting the constraint satisfaction problem corresponding to a musical decision into a binary decision diagram, which is a data
structure with special properties that make it possible to solve certain constraint satisfaction problems very efficiently and in a strictly bounded length of time.

As described so far, the ADTK, despite its modelling capabilities being unproven, satisfies our requirements for methods to support end user design of arrangement-level musical decision making (see Section 1.6.1). First, it is a general purpose method (requirement 1), at least insofar as there are no aspects specific to particular knowledge representations (i.e. representations of musical data). Second, it can be used for designing generative behaviour and interactive behaviour (requirement 2), the latter because association rules can describe dependencies between variables arising from a musician’s performance, and variables under an agent’s control (i.e. a musician can influence an agent’s decisions). Third, it requires no expertise in conventional computer programming (requirement 3); it has a point-and-click style graphical user interface featuring only standard elements such as buttons and drop-down menus. Finally, it can perform arrangement-level musical decision making in real time (requirement 4).

Thus, it remained to characterise the variety of arrangement-level musical decision making behaviours that could be modelled using ADTK agents. Furthermore, should the software prove capable of effectively capturing a wide variety of behaviours, the following additional questions were raised:

1. Does the requirement to select and configure custom variables along with parameters relating to association rules and VMMs, make the ADTK too challenging or unintuitive for musicians, in general, to use?

2. If so, are there preset configurations of custom variables and other parameters that might be supplied with the software to enable common musical patterns to be captured more easily?

3. Is the ADTK useful for developing musical agents for experimental interactive music systems similar to those described in Section 1.3?

4. What are the challenges involved in using the ADTK for these purposes and
5.1. Outline of the Research Related to the Agent Designer Toolkit

how might they be mitigated?

We refer to selecting custom variables and setting other model parameters collectively as designing the learning configuration for a musical agent. Based on early experimentation with the software, we assumed that broadly the answer to the first of these questions is affirmative, that is, the requirement to design a learning configuration in many cases does significantly reduce the accessibility of the software. While this is an important issue, we still considered the ADTK to be a substantial development in making the design of musical agents accessible to non-programming end users (we return to this below). That being so, in addition to characterising the modelling capabilities of the ADTK, we focussed on questions 2, 3 and 4, in the list above.

To address these questions, we conducted two studies. In the first study, we invited electronic music practitioners to submit compositions created in Max or Ableton Live. Specifically, they were asked to submit entire Live sets or Max patches with which they could perform live, along with recordings of the control data associated with sets of example performances (the latter, in order to train agents). For each participant, a set of agents was designed and each agent was used to generate new performances of the composition. The participant was then asked to give feedback on the agent performances. Some of the agents were designed by hand by the experimenter, and others were designed according to pre-determined sets of steps. Thus, it was possible to characterise the modelling capabilities of the ADTK, when used by an ‘expert’, and also to evaluate potential preset learning configurations (to address question 2 above). In addition, over the course of manually designing agents for each participant, we identified a number of promising techniques for using the features of the ADTK to capture common musical patterns. These contributed further to question 2, since such techniques could form the basis of additional or improved preset learning configurations.

The second study comprised five case studies in which the author collaborated with various artists to develop experimental interactive and generative music
systems. This allowed us to characterise the use of the ADTK in practise for collaboratively developing performance-ready systems. This study contributed substantially to questions 3 and 4 above.

The assumption, made above, that the requirement to design the learning configuration significantly reduces the accessibility of the ADTK, was explored in a third study focussing on the usability of the software. This provided a much more refined understanding of the usability issues introduced by custom variables and other parameters of the learning configuration. In addition, it helped to frame the issues identified in the second study.

Our view that the ADTK constitutes a substantial development in making the design of musical agents accessible to non-programming end users arises not least from the fact that it removes all need for conventional computer programming, but also for developing algorithms or heuristic schemes to determine agent behaviour. The studies outlined above provided additional support for this view. As well as demonstrating the modelling capabilities of the ADTK, the first study demonstrated the promise of certain preset learning configurations for capturing a variety of musical behaviours. This has important implications since such presets allow agents to be created simply by following sequences of pre-determined steps (these could in future be embedded into the software). In addition, the second study highlighted the ease and speed with which agents could be created for use in experimental music performances. As well as showing the promise of the ADTK in these regards, both studies resulted in concrete proposals for improving the modelling capabilities of the software; for developing and improving a selection of preset configurations; and for improving the ways in which the software can provide additional support for the design process.

We conclude this section with a brief outline of how our report on the research synopsized above is distributed across this chapter and the five that follow. In this chapter, we present a detailed rationale for the design of the ADTK, and an overview of the software itself (a walk-through of its use, including screenshots, is given
in Appendix A). This addresses research question III-(i) outlined in Section 1.6.2.
This is followed in Chapter 6, with the details of the implementation of the ADTK,
in particular, the real-time decision making components (research question III-(ii)).
Then, in Chapter 7, we present the first of the three studies mentioned above (research
question III-(iii)). In Chapter 8, we present the set of case studies that comprised
the second study mentioned above (research question III-(iv)). In, in Chapter 9,
we present the usability study alluded to above (research question III-(v)). Finally,
in Chapter 10, we draw together the results of the two studies and make specific
proposals for future work.

5.2 Outline of this Chapter

The remainder of this chapter is structured as follows. In Section 5.3, we examine
three electronic music systems, each corresponding to a particular composition that
is performed by sequencing and combining relatively short musical patterns, in addi-
tion to related activities such as switching audio effects on and off (i.e. arrangement-
level musical decision making). In particular, we study the data corresponding
to example performances with these systems in order to identify important mu-
sical patterns that we would require musical agents to be able to capture. We then
present the machine learning components of the ADTK in the form of a design ra-
ionale informed by our analyses of the data from the three electronic music systems
(Section 5.4). This is a detailed account of how we selected the machine learning
algorithms (VMMs and association rule learning) and combined their output to form
a single model of musical performance, before extending the model to capture more
sophisticated musical patterns (this involves the introduction of custom variables,
referred to above). We then give an overview of the ADTK software that supports
the design of these models and its integration into two popular electronic music
performance platforms: Max and Ableton Live (Section 5.5). This chapter concludes
by considering where the ADTK sits relative to current research in computer music
5.3 Analysis of Performance Data from Three Music Systems

In this section, we describe three custom systems for the performance of electronic music. They were used to gather preliminary data to inform the design of the ADTK. Each one is performed with at the arrangement level, that is, by selecting the instruments and the patterns to play, as well as choosing audio effects and setting the related parameters. We examine the control data recorded during performances with each system, with the hope of gaining insights into the types of patterns that a PBE system must be capable of learning.

Our intention was to acquire example performance data exhibiting contrasting musical styles and contexts. Therefore, the three systems were created separately by three different musicians—the author and two colleagues—and each is relevant to its creator’s own practise. In addition, a different input device was used to perform with each one. Finally, a different number of training examples were collected for each system, with ten examples being collected for the first, three for the second, and only a single one for the third.

Each of the three systems was designed for use by a single performer. This means that musical agents created to perform with them would be generative, rather than interactive (i.e. it would perform ‘solo’, without influence from a musician; see Section 1.1). This was partly for practical reasons, since it requires only a single musician to create example performance data with a system. However, our assumption is that in general the types of performance data patterns will be similar among generative and interactive systems.

The following three subsections are structured such that we first present the electronic music system and then discuss the example performance data that we recorded. We introduce the term *stylistically salient pattern* to refer to a pattern (Section 5.6).
or behavioural characteristic that a musical agent should reproduce so that its performance can be considered consistent with exemplar performance data. For each set of example performance data, we identify a selection of stylistically salient patterns and highlight the difficulty of differentiating these from other patterns that arise in the example data by coincidence and are not intrinsic to the performance style.

### 5.3.1 The Montreal System

The Montreal system was implemented by the author in Max and it is intended for performance of a particular electronic music composition. Like the Ensemble for Two Players system described in the previous chapter, the composition follows a repeating harmonic structure (the form is 16 bars long in this case). The musician’s performance activities include (i) switching on and off the different sound generators, each of which is controlled at a lower level by pre-composed sequences and probabilistic processes; (ii) changing the parameters of these low-level probabilistic processes; and (iii) setting the parameters of specific audio effects, and the signals being routed to them.

The graphical user interface of the Montreal system is shown in Figure 5.1 and described below.

**Section A** of the interface comprises a set of 11 binary-valued variables (check boxes) used to turn on and off the instruments and audio effects. There are nine instruments and two audio effects (labelled Through strings and Convolver; see below).

**Section B** of the interface includes an integer-valued variable referred to as the D. M. preset (drum machine preset), that changes the probabilities parameterising a probabilistic rhythm generator that controls the patterns being played by the percussion instruments. The D.M preset variable has three values corresponding to (i) a minimal repetitive rhythm, (ii) a more
embellished, vigorous rhythm, and (iii), a complex rhythm exhibiting
greater intensity than either of the others.

Section C of the interface comprises the routing matrix which comprises a bank of
84 binary-valued variables that control the routing of signals from the
instruments to the audio effects (see below).

Section D of the interface contains an integer-valued variable, Num conns (number
of connections), associated with the Through strings effect (again, see below).

The Montreal system has quantization functionality similar to Ableton Live. It allows
the performer to create a queue of parameter changes during one repetition of the 16-
bar form such that the changes only actually take place as the next repetition begins.
In this way multiple changes can occur simultaneously at the start of each form, but
changes cannot occur between the start of one form and the start of the next. Since
the 16-bar form lasts around twenty seconds, musical decision making with this
system is carried out on a time scale somewhat longer than typical arrangement-level
musical decision making. However, the musical structures that result (see below)
are quite general and relevant to the discussions which follow.
The routing matrix is used to control the routing of signals between the instruments and audio effects. Its 84 binary-valued parameters are grouped into 14 rows of 6 columns. The 14 rows correspond to the outputs of the 11 different instruments and effects (three of these have stereo outputs, and therefore have two rows associated with them). The 6 columns correspond to the inputs of two audio effects: Two columns correspond to the inputs of the Convolver effect (labelled ‘c1’ and ‘c2’ in Figure 5.1) and four correspond to the inputs of the Through strings effect (labelled ‘s1’ to ‘s4’ in Figure 5.1). Thus, when the variable corresponding to the matrix location at row 4, column 5 (with the row and column numbering beginning at 1) has a value of 1, then the output of the Snare Drum instrument is sent to the third input of the Through strings effect (this is one of the connections shown in Figure 5.1). These effects are described in the following.

The Convolver effect has two inputs. It uses the signal arriving at one input to modify that arriving at the other. Despite the name of the effect it does not do this by way of a standard convolution. Instead, it derives frequency-dependent amplitude envelopes from the signal arriving at its second input (‘c2’) and applies them to the signal arriving at its first input (‘c1’). One way to use the Convolver effect is to route the signal from an instrument producing a continuous pitched sound to the first input and the signal from a percussion instrument to the second input so that the rhythm of the percussion instrument is superimposed onto the sound of the pitched instrument.

The Through strings effect is a bank of six resonant filters that resonate at particular musical pitches\(^1\). Signals from other instruments may be routed to the Through strings effect so that they excite the filters. For example, routing signals from a percussion instrument to the Through strings effect causes rhythmic musical chords to sound. The effect has four inputs, each corresponding to a different (and randomly changing) subset of the six resonant filters. The Num conns variable controls the number of connections being made between the four inputs to the effect and the six filters.

\(^1\)The name arises because the filters are based on the technique for synthesizing plucked string sounds due to Karplus and Strong [104].
Table 5.1: The control parameters of the ensemble for two players. In the Type column, categorical is abbreviated by Cat. and Ordinal is abbreviated by Ord. The symbols used to represent the variables are given in the fourth column. Those corresponding to the routing matrix, $p_{1-84}$, are numbered columnwise. This means, for example, that the 14 variables associated with routing signals to the first input of the Convolver effect (‘c1’) are denoted $p_{1-14}$.

Finally, a point relevant to later discussions is that without signals being routed to them, neither of the effects just described can produce any sound (whether they are switched on or not).

A summary of the variables used to control the Montreal system is given in Table 5.1. In this table, the Domain column indicates the values that the variable is allowed to take and the type column indicates whether the values have a natural ordering (Ordinal) or not (Categorical). The symbols in the fourth column are used in formulae given later in the chapter. Note that there are 97 user-controlled variables in the Montreal system, though as mentioned below, not all were manipulated during the creation of example performances with the system.

**Example Performances**

The control data associated with 10 example performances using the Montreal system are shown in Figure 5.2. Snapshots of the variable values were taken once at the beginning of each repeat of the 16-bar harmonic structure. Each example performance comprises between 8 and 11 snapshots. Only the 48 variables that were manipulated...
Pattern

| 1 | The value represented by the colour yellow in Drum machine preset variable is never followed by that represented by the colour red. |
| 2 | The percussion instruments (Bass Drum, Echo Snare Drum, Snare Drum and Closed Hi-Hat) are often on at the same time (having been introduced gradually or all together) and they often switch off simultaneously. |
| 3 | The Chord Instrument is rarely used and is never used for more than one repeat. |
| 4 | There is always at least one pitched instrument active. That is, one of the following is always switched on: Bass 1, Bass 2, Lead Melody, String Effect, Chord Instrument, Convolver Effect. (As long as there is a pitched instrument being routed to the c1 inlet of the Convolver, then its output is pitched.) |
| 5 | When the Convolver effect is switched on, there are always between one and four signals routed to each of its inputs (represented by the rows labelled M-C1-XX and M-C2-XX in Figure 5.2). |

Table 5.2: A selection of stylistically salient patterns observable in the example performances made with the Montreal system.

Pattern Data from Three Music Systems

A list of stylistically salient patterns is given in Table 5.2. These were arrived at through examination of the performance data along with introspection on the part of the performer (the author, in this case) about his manner of performing with the system. The list is not exhaustive. In particular, some very obvious features of the performance are omitted such as the absence of any periods during which no instruments are switched on.

In addition to these being representative of the patterns that should be reproduced by an agent that performs with the Montreal system, they are examples of types of patterns that we expect to frequently encounter in the data corresponding to arrangement-level decision making in musical performance. Specifically, the first and fifth patterns both arise from a common feature of loop-based electronic music: musical intensity may increase in a variety of ways, but it often decreases abruptly. This abrupt reduction of intensity is referred to as a breakdown [41, pp. 91-92]. Thus, since as described above the D. M. preset variable has three values (red = least intense, yellow = intermediate, blue = most intense), the transition from yellow (intermediate) back to red (least intense) is unusual. Similarly, while the percussive instruments
Figure 5.2: The training data for the Montreal system. In each example, there are 48 rows, corresponding to the 48 variables that were used to perform with the system. Each column corresponds to one repeat of the 16-bar musical structure. All but the DM Preset and Num Conns variables are binary-valued. In the case of the binary-valued variables, the colour white indicates a value of 0, meaning ‘off’ in the cases of the instruments and effects, and ‘not connected’ in the case of the routing matrix (see text); and black indicates a value of 1 meaning ‘on’ or ‘connected’. The Num Conns variable is an ordinal parameter ranging from 1-9 and is represented by different shades of green. The DM Preset variable is a categorical value with three values, 0 (red), 1 (yellow) and 2 (blue).
are introduced in variety of ways (one by one, or in groups or all together) they are usually removed (i.e. switched off) together. The third pattern relates to an instrument that is used intermittently for variety, but does not play an important structural role in the music. We speculate that many music systems include variables of this kind, that parameterise instruments, effects or other aspects of the system’s output. The fourth and fifth patterns relate to the dependencies between parameters in a music system. We envisage that many performance styles with a given music system are in part characterised by recurring combinations of instruments, effects or other parameter values, so such dependencies may frequently correspond to stylistically salient patterns.

Importantly, as mentioned above, the stylistically salient patterns above were identified with cognizance of the system and the style underlying the example performances. It is possible to analyse the data naively and to identify many other patterns that are not stylistically salient. For example, from the data alone, it might be observed that when the routing matrix variable $M-S3-BD$ has a value of 1 (i.e. connected), then $M-S2-HH$ has a value of 1 too. This is a pattern in the performance data and yet it is not stylistically required; in many cases, the precise configuration of connections only makes a subtle difference to the music and so the relation between these two variables is not important. Furthermore, many configurations of the routing matrix that are not seen in the example data would also have been musically appropriate. Clearly from the data alone it is not always possible to distinguish between the patterns that are relevant to the underlying musical style and those that are not, and we expect this to be generally true. We will revisit this consequence of the paucity of training data at the end of this section and in the next.

5.3.2 An Electroacoustic Improvisation System

The second system used to inform our design of the Agent Designer Toolkit was created by Ben Carey in the context of improvised electroacoustic music. At the time of its development, Carey was a musician and interactive music systems researcher at
Table 5.3: The control parameters of the Electroacoustic Improvisation system. The domain of each variable corresponds to the selection of clips in a given track, with the value $-2$ being used when no clip is playing. In the Type column, categorical is abbreviated by Cat. and Ordinal is abbreviated by Ord.

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Type</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crashes</td>
<td>${-2, 0, 1}$</td>
<td>Cat.</td>
<td>$p_1$</td>
<td>Clip selection for the Crashes track</td>
</tr>
<tr>
<td>BassGrain</td>
<td>${-2, 0, 1, 2}$</td>
<td>Cat.</td>
<td>$p_2$</td>
<td>Clip selection for the BassGrain track</td>
</tr>
<tr>
<td>Voice</td>
<td>${-2, 0, 1}$</td>
<td>Cat.</td>
<td>$p_3$</td>
<td>Clip selection for the Voice track</td>
</tr>
<tr>
<td>Noise</td>
<td>${-2, 0, 1, 2}$</td>
<td>Cat.</td>
<td>$p_4$</td>
<td>Clip selection for the Noise track</td>
</tr>
<tr>
<td>Keys</td>
<td>${-2, 0, 1, 2}$</td>
<td>Cat.</td>
<td>$p_5$</td>
<td>Clip selection for the Keys track</td>
</tr>
<tr>
<td>FM</td>
<td>${-2, 0, 1, 2, 3}$</td>
<td>Cat.</td>
<td>$p_6$</td>
<td>Clip selection for the FM track</td>
</tr>
</tbody>
</table>

the University of Technology, Sydney. The ElectroAcoustic Improvisation (EAI) system comprised an Ableton Live set containing a collection of sounds which could be combined at performance time. The collection included both synthetic and acoustic sounds of various morphologies, from long synthetic timbres to articulate, rough and granular textures. The idea was to use these materials as a palette of sound objects with which to perform an improvised electro-acoustic piece. For live performance, Ableton Live was controlled using the TouchAble software\(^2\) running on an Apple iPad.

A set of six variables are used to control the system. These select the sounds (clips) being played on each of six Ableton Live tracks (see Table 5.3). Due to the sound content, the music produced by this system cannot be described as having an identifiable tempo or time signature. However, the Live set was configured to play at a tempo of 120 beats per minute with a time signature of 4/4, and the triggering of new clips was quantized so that sounds only started and stopped at the boundaries between bars. This means that there is a 2-second quantization imposed on the introduction and removal of musical material.

**Example Performances**

The control data associated with three example performances are shown in Figure 5.3. Snapshots of the clips playing in each track were recorded at the beginning of each performance.

\(^2\)See: touch-able.com.
5.3. Analysis of Performance Data from Three Music Systems

Figure 5.3: The training data for the Electroacoustic system. There were six Ableton Live tracks: *Crashes* with 2 clips; *BassGrain* with 3 clips; *Voice* with 2 clips; *Noise* with 3 clips; *Keys* with 3 clips; *FM* with 4 clips. Each colour indicates a clip index (e.g. red is index 0), but there is no particular significance attached to any index, nor any connection between clips in different tracks that happen to share the same index.

In general, this system is suited to the production of slowly evolving textural music, emphasising the juxtaposition of different elements, and there are no strict metre- or tempo-related considerations. Nor are there any particular combinations of material that cannot be used simultaneously. We envisage that these are common characteristics in ambient and textural music and they are reflected in the list of stylistically salient patterns given in Table 5.4. The first pattern corresponds to the infrequent changes in the *FM* track. This material is suited to extended exposition so clips are left to play for longer durations. The second and third patterns relate to the overall dynamics of the performances. Performances begin and end with only one track playing and there are never more than four tracks playing at once. Thus, there is clear long-term structure, but the precise combinations of material are not stylistically important. The final characteristic listed in Table 5.4 describes to the occasional synchrony of the *BassGrain* and *Voice* tracks. We expect this to be a
The primary compositional element in most drum and bass music is the breakbeat, or break, a sampled drum pattern typically taken from an existing musical recording. An extremely widespread example is the ‘amen break’, taken from the track ‘Amen Brother’ by the Winstons [60]. A typical approach to composing drum
and bass begins with time-stretching or compressing the source breakbeats so that they match the tempo of the track. Then, derivatives of each source breakbeat are created by slicing up the originals in different ways so that during performance, longer-term drum patterns can be created by sequencing a breakbeat and its derivatives. In Ableton Live, this can be done for each source breakbeat by copying it into a number of different clips on a particular track, and setting different start and end loop points for each one. Thus, each drum instrument in the composition corresponds to a particular track in the Live set containing a single break and its derivatives.

For the DNB system, a Live set suitable for drum and bass performance was prepared as follows. Using a standard suite of breaks\(^3\), two drum tracks were composed, each with seven different variants of a source breakbeat. In each drum track, the first clip (index 0, positioned at the top of the track) had a standard structure, meaning that it provided a steady beat, generally following a standard 4/4 drum and bass pattern, whereas the remaining clips (those with indices greater than 0) were fills, meaning that they had a short loop or a different pattern. One of the two drum tracks (White Drum in the example data that follows) had a harsh filter effect included in the signal chain that could be used for more intense fills or as a high intensity section. In addition, two bassline tracks were composed, each with two different variants which could be sequenced in different ways to give further structure to the track. Finally, there was a track of acapella vocal samples that included a delay effect in the signal chain. The choice of sample (if any) and the activity of the delay effect could be manipulated independently.

In total there were seven control variables for the system. Five related to clip selection in Ableton Live and two were for toggling on and off the two effects mentioned above. As with the EAI system described in the previous section, clip triggering was quantized to the bar boundaries. Example performances were made controlling the Live set using a Novation LaunchPad\(^4\).

\(^3\)Downloaded from: junglebreaks.co.uk
\(^4\)See: novationmusic.com
Chapter 5. The Agent Designer Toolkit

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Type</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Drum</td>
<td>{-2, 0, ..., 7}</td>
<td>Cat.</td>
<td>(p_1)</td>
<td>Clip selection for the Blue Drum track</td>
</tr>
<tr>
<td>White Drum</td>
<td>{-2, 0, ..., 7}</td>
<td>Cat.</td>
<td>(p_2)</td>
<td>Clip selection for the White Drum track</td>
</tr>
<tr>
<td>Yellow Bass</td>
<td>{-2, 0, 1}</td>
<td>Cat.</td>
<td>(p_3)</td>
<td>Clip selection for the Yellow Bass track</td>
</tr>
<tr>
<td>Blue Bass</td>
<td>{-2, 0, 1}</td>
<td>Cat.</td>
<td>(p_4)</td>
<td>Clip selection for the Blue Bass track</td>
</tr>
<tr>
<td>W.D. Filter</td>
<td>{0, 1}</td>
<td>Cat.</td>
<td>(p_5)</td>
<td>Turn the White Drum Filter on/off</td>
</tr>
<tr>
<td>Acapella Vox</td>
<td>{-2, 0, 1, 2}</td>
<td>Cat.</td>
<td>(p_6)</td>
<td>Clip selection for the Acapella Vox track</td>
</tr>
<tr>
<td>Vox Delay</td>
<td>{0, 1}</td>
<td>Cat.</td>
<td>(p_7)</td>
<td>Turn the Vox Delay effect on/off</td>
</tr>
</tbody>
</table>

Table 5.5: The control parameters of the Drum and Bass system. The domains of five of the seven variables correspond to the selection of clips in a given track, with the value \(-2\) being used when no clip is playing. In the Type column, categorical is abbreviated by Cat. and Ordinal is abbreviated by Ord.

Example Performances

The control data associated with an example performance of the Drum and Bass system is shown in Figure 5.4. Snapshots were taken at the beginning of each bar to record the clips playing and the on/off state of the two effects. The example is 148 bars long.

As for the previous two systems, we list a selection of stylistically salient patterns of the performance data (see Table 5.6). The first of these relates to a straightforward dependency between the two bass tracks; the value of one affects the value of the other. We envisage this type of pattern arising in many different musical contexts in which it is inappropriate to juxtapose particular instruments or effects. Each of the three remaining patterns relates to the strict hypermetrical structure (i.e. ‘meter above the level of the measure’ [171]; see, e.g. [41, pp. 185-201]) of Drum and Bass music. The second describes the four-bar patterns observable in the two drum tracks. The third and fourth patterns relate to the sequencing of the W.D. Filter and Acapella Vox variables and its relation to the four-bar structure in the drum tracks. We expect that stylistically salient patterns in electronic music will frequently relate to a four-bar (or other) metrical structure in the performance data.

Finally, we note again that additional knowledge is required to identify the stylistically salient patterns in the performance data. In this case, extra information
### Pattern

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>There is only ever one bass playing at a time.</td>
</tr>
<tr>
<td>2</td>
<td>Each of the two drum tracks are structured in groups of four bars. Each four-bar pattern is one of the following: AAAB, AABB or AAAA, where A is a basic rhythm and B is a fill (occasionally a rest is used instead of a fill). Of these patterns, the first is the most common.</td>
</tr>
<tr>
<td>3</td>
<td>The filter effect used on the White Drum track is turned on for four-bar segments, aligned with the four-bar structure of the drums.</td>
</tr>
<tr>
<td>4</td>
<td>The vocal samples (Acapella Vox) are triggered in a less structured manner, but still frequently adhering to the four-bar structure of the music.</td>
</tr>
</tbody>
</table>

Table 5.6: A selection of stylistically salient patterns observable in the the example performances made with the Drum and Bass system.
is required particularly to generalise from the exhibited patterns. For instance, the four-bar structure of the drum tracks is quite evident, but it is unclear from the data alone whether the precise ordering of fills is important, or whether there are subtle relationships between fills in the different tracks (there are not).

5.3.4 Analysis Outcomes

The terms overfitting and underfitting were introduced in the context of machine learning in Section 2.2. A generative model of arrangement-level musical decision making that is overfitted to a training data set would tend to copy the training data set, rather than to emulate the style underlying it. Conversely, a generative model of the same type that is underfitted to a training data set would fail to reproduce the style underlying it. Generally, the problem of overfitting can be mitigated by increasing the size of the training data set. However, this is not true of underfitting, since it can be the case that the model is simply not complex enough to capture a particular pattern (e.g. modelling a curve with a straight line).

The analyses of the performance data reported in this section inform the design of the machine learning components in two ways. First, they give an indication of the types of patterns that we will require the machine learning algorithms to be capable of capturing. This helps ensure that the system can produce musical models of sufficient complexity (i.e. they do not underfit the training data). Second, the analyses highlight the need for mechanisms by which the learning algorithms can differentiate between stylistically salient patterns, and patterns that are not important (i.e. the need to prevent overfitting). The need for balancing these two concerns underlies much of the rationale, presented in the next section, for our design of the machine learning components of the ADTK.
5.4 A Rationale for the Learning Components of the ADTK

The learning components underlying the ADTK comprise two separate unsupervised machine learning algorithms and a variety of means by which the user can introduce additional musical knowledge distinct from that in the training data set. In this section, we describe these components and give a rationale for the selection of the machine learning algorithms and the design of the ways in which a user can interact with them. While there are formal and semi-formal methods for presenting software rationales (see, e.g. [39, Ch. 6]), what follows is a free-form account intended to outline the conceptual underpinnings of our design. In addition, we are concerned here primarily with the creation of models of arrangement-level musical decision making, the parameters of which will be trained by the learning components of the ADTK. We do not discuss the implementation of the software, or issues related to computational complexity or real-time computation. These topics are addressed in the next chapter.

5.4.1 Requirements of the Machine Learning Algorithms

The performance data examined in Section 5.3 comprises discrete multi-variate time-series. At first glance, there is a wide selection of machine learning methods for modelling such data, including hidden Markov models and probabilistic graphical models, more generally, as well as many others. However, we narrow the scope of this discussion by making the assumption that in general, the training data set will be insufficient to be used naïvely as input to any ‘black box’ machine learning algorithm. In other words, we assume that there is no machine learning algorithm that could, without any configuration of the learning parameters or pre-processing of the training data set, effectively learn the stylistically salient patterns in the performance data examined in Section 5.3.

This assumption of data paucity implies that for the machine learning to be
effective, additional information of some kind will be required from the user. We recall that in the envisaged typical use case for the ADTK, the user of the software is both the designer of the music system with which an agent is required to perform, and the creator of the training data set (see the aims listed in Section 1.6.1). Thus it seems reasonable to expect that the user will in general be in a position to identify the stylistically salient patterns in the training data set. The challenge in designing the ADTK is to allow the user to easily configure the machine learning system.

In general, there are three activities involved in configuring a machine learning system. They are,

1. choosing the machine learning algorithm(s);

2. setting the parameters of the machine learning algorithms; and

3. performing feature selection.

These are three of the stages comprising the standard machine learning workflow (see Section 2.2), and together they constitute choosing a suitable model for the data along with the details of how it will be trained.

For users with no expertise in machine learning these are challenging activities. In a study of the Wekinator—computer music software based on machine learning—Fiebrink found that such users rarely changed the learning algorithm from the one selected in the software by default, even though other options were available which in some cases would have been more effective [80, Ch. 5]. Similarly, it was rare for users to experiment with changing the parameters of machine learning algorithms or with selecting different features of the data to use as input [80, Ch. 5]. Fiebrink notes that ‘it was clear from students’ [i.e. participants’] comments that some of them did not understand how to use or interpret one or more of these components’ and while she speculates that novice users could be better educated via the user interface or through help materials, we believe that a requirement to undertake such education
could substantially deter users\textsuperscript{5}.

For the reasons given above, it may seem without promise, \textit{prima facie}, to develop a machine learning system requiring users to substantially engage with any of the three activities that comprise the standard machine learning workflow. However, we note that users of the \textit{Wekinator} were required to choose between advanced machine learning algorithms (e.g. artificial neural networks, decision trees and support vector machines) all of which are configured by setting the values of parameters that are opaque to non-experts. It may be possible to make these algorithms more accessible through user-interface innovations. However, we chose to explore an alternative approach, characterised by the use of using machine learning algorithms that present a lower conceptual barrier to users. Thus, we narrow our discussion of machine learning algorithms with the following requirements:

\begin{itemize}
\item \textbf{Accessibility}: An algorithm should plausibly present a low conceptual barrier to end users. This means that its parameters should be transparent (to the extent that is possible) to users who are not experts in machine learning.
\item \textbf{Graceful degradation}: An algorithm should not require large amounts of training data, or at least it should ‘gracefully degrade’ as the amount of data is reduced.
\end{itemize}

In the following, we identify two machine learning algorithms that meet these requirements.

\subsection*{5.4.2 Two Algorithms that Plausibly Meet the Requirements}

While it is often straightforward to characterise the performance of a machine learning algorithm with small training data sets, it is difficult to do so with respect to the extent to which it is accessible to end users. Most research related to the use of machine learning algorithms by end users has been conducted with the aim of

\footnotetext{5}{Our belief is that this is the case for new, unproven software. Obviously, many users undertake time-consuming and sometimes expensive training to become expert users of established professional music software.}
finding ways to avoid requiring users to choose machine learning algorithms, their parameters, or the data features to use as input (see the discussion of interactive machine learning in Section 2.2.2). Indeed, we could find no study in the literature assessing the accessibility of machine learning algorithms to end users.

Instead of making assertions about the accessibility of machine learning algorithms, we examine commercial music software packages to inform our selection. While music software rarely exposes machine learning algorithms to users, many packages make available techniques for the probabilistic generation of musical material. Frequently, these include

- the probabilistic sequencing of musical notes or other material using first-order Markov models, and
- the generation or manipulation of musical material using rules, sometimes involving indeterminate outcomes.

In the following, we argue that (i) the prevalence of these techniques is an indication that they are accessible to end users and (ii) though the machine learning algorithms associated with these techniques are less prevalent, they plausibly present a low conceptual barrier to end users.

Examples of software products that make Markov models and rules available for the generation and manipulation of musical material are shown in Table 5.7. We note that three of the products listed (Ableton Live, Cubase and Logic) have been listed among the top ten most popular music software packages at the time of writing [70], though of course, it is not possible to gauge the popularity of the relevant features of these packages. Nevertheless, this shows the widespread availability of these techniques and as such, it lends some credence to the notion that they are readily accessible to end users.

Each of the two techniques listed above involves the use of a model to generate musical data and in each case the model can be defined by hand (as is required in all of the software packages in Table 5.7 with the exception of Jam Factory), or learnt by
A user can choose the parameters of a first order Markov model (albeit from a limited set of pre-determined options) that can then be used to generate sequences of clips [2].

A Markov model is learned from note sequences played by the user and it is then used to generate new sequences [187].

The user defines Markov models by placing nodes in a 2D space and making connections between them [127].

Using the Logical Editor in Cubase a user can define probabilistic rules for the generation and manipulation of MIDI note data [164].

The Transform Window in Logic allows the user to define rules to manipulate MIDI note data [11].

Noatikl allows the user to explicitly introduce probabilistic rules to control musical output. To define a rule, the user specifies the conditions under which it will apply and the possible probabilities of various outcomes (see the creation of Harmony rules in [56]).

Table 5.7: Examples of the availability of Markov models and probabilistic rules in commercial music software.

Machine learning algorithms: Markov models are trained by calculating the required probabilities from a training data set, while rules can be learnt by association rule learning algorithms which learn implies rules (e.g. \( A \implies B \)), meaning that when event \( A \) occurs, \( B \) must occur too) from a training data set (further details are given below). Of course, that a model itself presents a low conceptual barrier to a user is no indication in general that the same will be true of the machine learning algorithm used to train it. However in the cases of Markov models and association rule learning, we argue that this plausibly is the case.

The algorithm required to train a first-order Markov model is trivial and does not require configuration (see Section 2.3). However, as discussed below, higher-order Markov models would be required to learn many of the stylistically salient patterns identified in Section 5.3. This means that a user would be required, at most, to choose the order of any Markov model used. While this is not required of users of the software products listed in Table 5.7, our supposition is that the order of a Markov model is a transparent parameter with which users could fruitfully engage, since high orders will usually result in the generation of data that resembles the training data set more closely than lower orders. A disadvantage of higher-order
Markov models is that they do not ‘gracefully degrade’ (again, see Section 2.3). However, they can easily be replaced by variable order Markov models (VMMs) which can be used with much smaller training data sets. VMMs do not require any additional configuration.

An overview of association rule learning algorithms is given in Appendix B. Here, and elsewhere in this section, we introduce the essential aspects that are relevant to this rationale. To configure association rule learning (ARL) algorithms, a user typically must select (i) the variables among which rules will be sought; (ii) the minimum certainty required for a rule to be accepted (the required confidence); (iii) the minimum prevalence of a rule describing how often it arises in the training data set (its required support); and (iv) the maximum number of variables that can be involved in a rule (again, details are given below). The first and fourth of these parameters are transparent and have a large effect on the set of rules that are found. Our expectation is that the second and third parameters are less important with respect to the effective learning of stylistically salient patterns, and in any case we suppose that the challenge presented by their selection is comparable to that associated with the direct specification of harmony rules in Noatikl, for example [56].

To sum up, our position is that broadly, VMMs and ARL algorithms plausibly present a low conceptual barrier to users aiming to design agents for arrangement-level musical decision making. Thus, at the outset we selected these algorithms to explore their potential as the basis of the learning components of the ADTK. In the following, we begin this exploration by discussing separately the application of VMMs and ARL techniques to learning the stylistically salient patterns in the training data sets presented in Section 5.3.

### 5.4.3 Variable-Order Markov Models

VMMs are most readily applied to one-dimensional sequences of discrete-valued variables. A variety of methods have been studied for extending them to modelling sequences of multi-dimensional variables (e.g. the multiple viewpoint methods
5.4. A Rationale for the Learning Components of the ADTK

Figure 5.5: Illustrations of performance data generated by training a separate Markov model for each music system variable. (a) Generated performance data for the Montreal system using first-order Markov models; (b) Generated performance data for the DNB system using seventh-order Markov models; (c) Generated performance data for the EAI system using 20th-order Markov models.

described in Section 4.1.1). However, since our selection of Markov models was predicated on their use in commercial music software packages, and in these packages Markov models are usually used to model only one-dimensional variables, we restrict this discussion accordingly.

In the following, we describe a range of properties of VMMs as they relate to learning patterns such as those identified in Section 5.3. In order to provide illustrative examples, we modelled the Montreal system performance data using first-order VMMs; (ii) the DNB system performance data using seventh-order VMMs; and (iii) the EAI system performance data using 20th-order VMMs (see Figure 5.5). (As mentioned in Section 2.3.2, an nth-order VMM is one in which the order used is limited to n.) These models will be referred to throughout.
Low-order VMMs Generally Capture Low-order Statistics

Low-order VMMs are suitable for capturing structure on a short time scale. For example, a first-order Markov model is sufficient to capture the stylistically salient pattern related to the DM Preset variable of the Montreal system (pattern 1, Table 5.2). With a model trained on the Montreal training data set, the probability of going from ‘yellow’ to ‘red’ is zero. Similarly a first-order Markov model can capture the infrequent use of the Chord Instrument in the Montreal system, and that when it is used, the Chord Instrument is only left on for one repeat (pattern 3, Table 5.2). Both of these patterns are illustrated by the generated data shown in Figure 5.5(a).

A first-order VMM can capture the low-order statistics of sequences of variable values, however when the sequences in the training data set are very short, there are ‘end effects’ that can be significant. We explain this using the Bass 1 variable of the Montreal system for illustration. The transition matrix learnt for the Bass 1 variable (this can be derived by examining the trained VMM), is:

<table>
<thead>
<tr>
<th></th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>62/70</td>
<td>8/70</td>
</tr>
<tr>
<td>off</td>
<td>1/2</td>
<td>1/2</td>
</tr>
</tbody>
</table>

where the rows correspond to the ‘current’ state, and the columns, to the ‘next’ state. Among other characteristics, this broadly describes the high likelihood that when the Bass 1 instrument is switched on, it is more likely to remain on than switch off. However, according to Equation 2.6, once it turns on, the expected number of forms for which Bass 1 will remain on is 7.75. In contrast, the mean number of forms for which Bass 1 remains on after being switched on, as calculated from the training data set, is approximately 5.63. The discrepancy between the expected duration associated with the Markov model and the mean calculated from the data is due to the fact that three of the example performances (Examples (c), (f) and (g) in Figure 5.2) end with Bass 1 switched on. This means that there are three instances in which the period for
which the instrument remains on is truncated (by the performance ending) without a corresponding on → off transition being accounted for in the Markov transition probabilities (this is the ‘end effect’ alluded to above). If the Markov transition probabilities are adjusted to account for these three omissions (i.e. the top row of the matrix above becomes: \( \frac{62}{73}, \frac{11}{73} \)), the expected duration associated with the Markov model becomes identical to that calculated from the training data. While this ‘end effect’ is worth noting, it is only significant where very short training sequences are used, and in all other cases, first-order (and higher-order) VMMs do capture the low-order statistics of the training data accurately.

**The VMM Order is Limited by History Length**

A seventh-order VMM was chosen to model the performance data for the DNB system, because this is the minimum order required to capture the four-bar groupings characteristic of the Blue Drum track. This is because seven bars is the longest duration for which the pattern does not change in the Blue Drum track training data. Thus, to ensure that a fill is played on the eighth bar, a seventh-order VMM is required.

However, in the example shown (Figure 5.5(b)) there is misalignment between the four-bar groups in different tracks (e.g. Blue Drum and White Drum; cf. pattern 2, Table 5.6). The four-bar groups of the Blue Drum track are shifted by two bars (i.e. the clips are structured in groups of four but the regular fill—indicated by the clips that are not orange in colour—occurs on the second bar, rather than in the fourth one). In a similar way, the four-bar groups of the White Drum track are shifted by one bar.

The misalignment of the four-bar groups of clips arises for two reasons. In the case of the Blue Drum track, it is because the order at which a VMM operates is limited not only by the maximum order (seven in this case), but also by the history-length. This is described in detail in Section 2.3.2 and here it accounts for the first fill occurring on the second bar, rather than the fourth one (from that time onwards, the
four-bar groupings are reproduced, but the are shifted by two bars). As described in Section 2.3.2, this problem can be solved by introducing an ‘invisible’ start state.

However, the misalignment of the four-bar groupings in the White Drum track cannot be solved by the same method. This is because at the beginning of the training example, there is a period of 40 bars for which the White Drum track is inactive. To ensure that when the White Drum becomes active, it does so such that the four-bar groupings are correctly aligned, a 40th-order VMM would be required. While this is feasible, it means that sequences generated for the White Drum track would precisely copy that in the training data set.

**High-Order VMMs Tend to Duplicate the Training Data Set**

A number of examples of the problem whereby a high-order model duplicates the training data set (or substantial portions thereof), are found in the data shown in Figure 5.5. In the sequence generated for the Blue Drum track (Figure 5.5(b)), the pattern of fills duplicates portions of the training data set. For example, if the ‘red’ fill is used, then it is never followed in the next four-bar group by the ‘brown’ fill and this can never happen because it never occurs in the training data set. This can be seen as an overfitting problem (see Section 2.2) due to the seventh-order VMM used, however as noted above, the order was chosen because it is the minimum order required to capture the structure of the Blue Drum track. Thus, in this instance, due to the limited amount of training data available, it is not possible to learn the stylistically salient pattern without either underfitting or overfitting the model to the training data set.

Other examples of overfitting can be seen in the performance data generated for the EAI system (Figure 5.5(c)). The 20th-order VMM duplicates large sections of the training data set. This is particularly evident in the patterns of ‘red’ clips in the Crashes track and that of ‘red’ and ‘green’ clips in the Voice track.
VMMs can get ‘stuck’ on a particular value

As described in Section 2.3.2, it is possible for a training data set to give rise to a VMM which will get ‘stuck’ on a particular value when it is used to generate a sequence. Also outlined is a simple solution to this problem: the VMM is trained, not on the original sequence of values, but on a ‘looped’ sequence comprising two copies of the original sequence concatenated together. This problem cannot arise with any of the training data sets discussed in this chapter, however, it would clearly be beneficial to include the solution in any system intended to train VMMs using small training data sets.

Summary

We have illustrated the following properties of VMMs that are pertinent to their use in a PBE system for designing arrangement-level musical decision making:

- Low-order VMMs can capture short temporal structure and low-order statistics, such as the expected duration for which a variable value will remain unchanged. However, short training data sequences can give rise to ‘end effects’ which may significantly distort the statistics of generated sequences.

- The order at which a VMM operates is limited by history-length. In some cases this can be problematic but the problem can be mitigated by introducing ‘invisible’ start states, as described in Section 2.3.2.

- High-order VMMs can readily capture temporal structure on a longer time scale. However, overfitting can arise in cases where long temporal structures must be learnt from a small amount of training data.

- Certain VMMs can get ‘stuck’ on a particular value. Such VMMs can be avoided by using ‘looped’ sequences in the training data set (see above and Section 2.3.2).
Finally, by definition, independently-operating VMMs cannot reproduce dependencies between variables and this means that many stylistically salient patterns cannot be captured (e.g. pattern 1, Table 5.6—only one bass instrument may play at a time—is violated in the generated data shown in Figure 5.5(b)). However, association rule learning techniques can model such dependencies and in the following sections, we discuss their capabilities and their combination with VMMs to form models of arrangement-level musical decision making.

### 5.4.4 Association Rule Learning

Association rule learning has not, to our knowledge, been previously used to discover patterns in musical data. As briefly mentioned above, association rule learning refers to a body of techniques for learning rules that describe the dependencies in a data set. Specifically, given a set of observations of the values of a set of variables ARL techniques can be used to search for dependencies between the variables. Thus, by using as observations the instantaneous values of the music system variables at each decision point in the training data set, dependencies between the variables can be discovered.

Dependencies are discovered in the form of association rules, which are **implies rules** of the form

\[ A \implies B, \]

where \( A \) is an event referred to as the *antecedent* and \( B \) is an event referred to as the *consequent*. Also mentioned above are the parameters that are typically required to configure an ARL algorithm. They are,

1. The set of variables among which rules will be sought (in the following, we will refer to such a set as a rule group);
2. The minimum certainty required for a rule to be accepted (the minimum confidence);
3. The minimum prevalence of a rule which describes how often it arises in the
training data set (the minimum *support*); and

4. The maximum number of variables that can be involved in a rule. This is referred to as the maximum *itemset size*.

Further details about association rule learning are given in Appendix B. In the following we illustrate the application of association rule learning to discovering stylistically salient patterns in the training data sets presented in Section 5.3. In addition, we discuss how configuration of the four parameters listed above can influence the rule learning process, and in particular, how it can control the extent to which the learnt rules overfit or underfit the training data set.

**The Relevance of Association Rule Learning to the Discovery of Stylistically Salient Patterns**

Association rule learning is capable of discovering stylistically salient dependencies in the *Montreal* training data set. An association rule learning algorithm (the *Apriori* algorithm, see below) was applied to a rule group containing ten of the eleven music system variables that switch on and off the instruments and effects of the *Montreal* system (these are the switches marked ‘A’ in Figure 5.1; the *Chord Instrument* variable is omitted since it is very rarely used). The minimum confidence was set to 100% (i.e. only rules consistent with every observation were accepted); the minimum support was set to 5%; and the maximum number of variables in a rule was set to three. In total, 58 rules were found. Three examples are given here:

\[
\text{Bass Drum Off AND Snare Drum Off } \Rightarrow \text{ Echo Snare Off,} \quad (5.1)
\]

\[
\text{Through Strings Off AND Echo Snare On } \Rightarrow \text{ Hi-Hat On,} \quad (5.2)
\]

\[
\text{Bass 2 On AND Lead Melody Off } \Rightarrow \text{ Convolver On,} \quad (5.3)
\]

The first of these, for instance, describes the dependency of the *Echo Snare* instrument on the *Bass Drum* and *Snare Drum* instruments: if the latter two are off (i.e. they are inactive), then the *Echo Snare* must be off too. These rules are clearly relevant
Chapter 5. The Agent Designer Toolkit

to the stylistically salient patterns identified for the Montreal system. For example, Rule 5.1 describes a dependency amongst the variables controlling the percussion instruments and as such, it is related to pattern 4, Table 5.2.

Association rule learning can also discover rules corresponding to the stylistically salient patterns of the performances with the DNB system. As a straightforward example, the Blue Bass and Yellow Bass variables were placed in a rule group for ARL analysis. The following rules were found:

\[
\begin{align*}
\text{Yellow Bass plays clip 0} & \implies \text{Blue Bass is Off} \quad (5.4) \\
\text{Yellow Bass plays clip 1} & \implies \text{Blue Bass is Off} \quad (5.5) \\
\text{Blue Bass plays clip 0} & \implies \text{Yellow Bass is Off} \quad (5.6) \\
\text{Blue Bass plays clip 1} & \implies \text{Yellow Bass is Off} \quad (5.7)
\end{align*}
\]

Together, these correspond exactly to pattern 1, Table 5.6, whereby there is only ever one Bass instrument playing at a time.

**Controlling Rule Learning**

The rule learning parameters can be used to control the manner in which the rules learnt model the training data set. Consider the ten binary-valued variables of the Montreal system, amongst which rules were sought in the example above. These variables can take on 1024 (= 2^{10}) different combinations of values but only 62 unique combinations are observed in the training data set (see Figure 5.2). If a set of rules was discovered and only these 62 combinations were consistent with it, then this set of rules is likely to overfit the training data set. In contrast, if a set of rules was discovered and 1000 combinations were consistent with it, then this set of rules is likely to underfit the training data set.

In the example above, 58 rules were discovered describing the dependencies between the 10 binary-valued variables of the Montreal system. There are 264 unique combinations that are consistent with this set of rules. These include the 62 observed
configurations and 202 others. If the maximum itemset size is increased to four, a more restrictive set of rules is discovered with which only 78 configurations remain consistent. These comprise the 62 observed ones and 16 others. When the maximum itemset size is raised to six, the discovered rules model the training data set exactly; only the 62 observed configurations are consistent with the rules discovered; this is the overfitting scenario alluded to above. Thus, the maximum item set size can influence the extent to which the discovered rules overfit or underfit the training data set.

The minimum support can also be used to manipulate the set of rules that is discovered. In particular, raising the minimum support means that only rules that arise more frequently in the training data set will be accepted. Re-running the example above with the minimum support increased to 15%, the number of configurations consistent with the discovered rules increases to 298, comprising the 62 observed configurations and 236 others.

Conversely, lowering the minimum confidence increases the number of rules found since rules do not have to be true all of the time in order to be accepted. Re-running the example above with a minimum confidence of 90% reduces to 46 the number of configurations that are consistent with the discovered rules. Moreover, only 32 of these are found in the training data set and 14 are not, leaving 30 observed configurations inconsistent with the discovered rules. Thus, the effect of lowering the minimum confidence required is not only to restrict the number of configurations that are consistent with the discovered rules, but also to bias the learning in ways that may be difficult for a user to predict. Lowering the minimum confidence further can give rise to rule sets with which there are no consistent combinations (i.e. certain rules contradict each other, leaving no combination of variable values consistent with all the rules). Re-running the example above with a minimum confidence of 60% gives this result.

Finally, the choice of rule group determines the variables amongst which rules will be sought. In discussing the training data for the Montreal system, it was noted
that a naïve analysis of the training data set might conclude that two particular connections of the routing matrix were always used together, corresponding to the rule:

$$M \text{-} S3 \text{-} BD = 1 \implies M \text{-} S2 \text{-} HH = 1. \quad (5.8)$$

Indeed, if the routing matrix parameters were included in a rule group and analysed for association rules, it is likely that this rule would be discovered. However, by choosing only rule groups that contain variables amongst which stylistically salient dependencies might be found, the discovery of this rule could be avoided.

**The Limitations of Association Rule Learning**

There are significant limitations on the types of dependencies that can be discovered. For example, the requirement that at least one signal be routed to each input of the **Convolver effect** when it is switched on (pattern 5, Table 5.2) cannot be discovered by association rule learning. Rules such as

$$\text{Convolver} = 1 \implies p_1 = 1 \text{ OR } p_2 = 1 \text{ OR } \ldots \text{ OR } p_{14} = 1 \quad (5.9)$$

would be required. (The variables $p_1 \ldots p_{14}$ correspond to the elements of the routing matrix that route signals to the first input of the Convolver effect; see Table 5.1.) Such rules, involving logical ‘or’ clauses are referred to as generalised association rules. The **Apriori** algorithm [3], used for the illustrations above, is not capable of discovering generalised association rules. There do exist other rule-finding techniques that could discover rules of this form, such as the patient rule induction method [83] and the use of classification and regression trees described in [34]. However, they are not exhaustive; they are not guaranteed to find all relevant rules, nor even ‘optimal’ sets of rules [92, p. 501].

A more important limitation of ARL, perhaps, is the difficulty in learning time-dependent rules, that is, context-dependent dependencies between variables. For instance, rules describing the presence of just one track to begin each performance
of the EA system could not be discovered (pattern 2, Table 5.4). Nor could rules
describing the occasional synchronicity of the BassGrain and Voice tracks in the same
system (pattern 4, Table 5.4).

**Summary**

We have demonstrated the application of ARL to the discovery of stylistically salient
patterns in arrangement-level performance data. In particular, we have demon-
strated that

- Stylistically salient rules can be discovered; and
- Rule discovery can be controlled by adjusting the parameters of the ARL
  algorithm.

However, ARL techniques have significant limitations. While the use of generalised
association rule learning algorithms may in part mitigate these, such algorithms
are not ideal since they are not exhaustive. Thus in the following, we discuss the
amalgamation of VMMs and (non-generalised) association rules to form a model of
musical performance, and then propose another approach by which more sophisti-
cated and time-dependent relationships can be captured.

**5.4.5 Combining Markov Models and Rules to form a Model of
Music Performance**

At this point, we have discussed Markov models and association rule learning
techniques with respect to their potential for learning the stylistically salient patterns
underlying the training data sets presented in Section 5.3. Significant limitations
were identified for each technique, specifically,

1. VMMs can give rise to inappropriate combinations of variable values (since
dependencies between variables are not modelled);

2. high-order VMMs tend to overfit the training data set when it is limited in size;
3. ARL techniques cannot capture relationships involving logical ‘or’ clauses; or

4. context-dependent relationships.

Observing that Markov models are most readily applied to modelling the temporal structure associated with individual variables, and in addition, that association rules are most suited to modelling the relationships between variables, our approach was to combine the two to form a single model of music performance not characterised by limitation 1 above, and then to explore ways to extend this model to remove limitations 2-4. In the following, we describe how VMMs and association rules may be combined to form a simple model of arrangement-level musical decision making.

Together, a set of association rules defines a set of combinations of variable values that are ‘allowed’ (i.e. they do not break any rules). In cases where the independently-operating VMMs generate a combination of values that is not found in this set—it is not allowed—there are broadly two options available. Either (i) new values are repeatedly generated by all VMMs until an allowed combination arises, or (ii) certain VMM-generated values are retained and others are rejected, and some other procedure is used to generate new values to replace the rejected ones. Since the first option would take an unpredictable length of time to compute the second one must be taken. The questions then concern how to choose which VMM-generated values are retained, and, how to choose new values to replace the rejected ones.

Setting aside questions of computational burden, it would be possible to perform an analysis to find the smallest set of VMM-generated values that needs to be replaced, thus retaining the largest set. However, there is no reason to assume that the variables are of equal importance with respect to the style underlying the training data set. A simpler method that does not embody this assumption is to allow the user to place variables in a ‘priority’ order (i.e. the user ranks the variables in order of their importance).

With a priority order in place, a set of values that is consistent with the rules can be found as follows. For each variable, starting with the highest-priority one and
continuing in order of decreasing priority, the following steps are taken:

1. Find values for the variable that are *not* consistent with allowed combinations.

2. Modify the probability distribution from which a new value will be drawn by the VMM (this depends on the history of variable values) so that these ‘not-consistent’ values have probabilities of zero.

3. If this results in an invalid probability distribution with no non-zero entries, then reduce the order of the VMM by one and return to step 2. Otherwise, draw a value from the modified probability distribution.

4. Prune the list of allowed combinations so that only combinations that include the newly-drawn value are retained.

Steps 2 and 3 must produce a suitable value eventually (and in a bounded amount of time), since eventually the VMM will be reduced to a 0th-order one and the probability distribution associated with a 0th-order VMM must contain non-zero probabilities for all values in the domain of the variable (see Section 2.3). Thus, by following this procedure for each variable in turn, new values will be found for all variables controlled by the agent.

**5.4.6 Extending the Model with Custom Variables**

In the previous section we described a generative model of arrangement-level musical decision making, combining VMMs and association rules. Here, we discuss its extension to address limitations 2-4 listed at the beginning of the previous section. We begin by noting that the modelling capability of association rules can be increased by including more variables such as

- the values of music system variables at previous decision points, and

- ‘dummy’ variables to represent potentially useful quantities (e.g. variables that encode events of the form ‘X or Y’ [92, Ch. 14]).
Including the values of variables at previous decision points would allow patterns in the changes of variables to be discovered. For example, one of the stylistically salient patterns for the Montreal system describes the percussion instruments frequently turning off at the same time (as part of a breakdown; pattern 2, Table 5.2). If the previous values of the Snare Drum variable were included, the following rule could be found by association rule learning:

\[
\text{Snare Drum Previous On AND Snare Drum Off } \implies \text{Closed Hi-Hat Off} \quad (5.10)
\]

In other words, ‘when the Snare Drum turns off, the Closed Hi-Hat must be off’, thus, if the Closed Hi-Hat had been on, it would have to turn off too. In addition, dummy variables would allow the discovery of generalised association rules involving logical ‘or’ clauses such as that in Equation 5.9.

However, it would be difficult to automate the discovery of stylistically salient patterns using new variables such as those suggested above. For example, the number of dummy variables that might give rise to stylistically salient rules being discovered grows combinatorially with the number of binary variables required to encode all of the music system variables (a dummy variable is required for every possible union of events). Moreover, we assume that including a complete set of such dummy variables would result in the discovery of many rules that arise only by chance in the training data set, and do not correspond to stylistically salient patterns (i.e. overfitting would result). Instead of automatically pre-coding an exhaustive set of dummy variables, we chose to allow the user to augment the model by choosing those that correspond to real stylistically salient patterns in the training data set.

We introduce the term custom variable to refer to a (dummy) variable created by the user, the value of which is a function of the values of a subset of the music system variables. For example, to ensure that signals are routed to the Convolver effect in the Montreal system (pattern 5, Table 5.2), a user could introduce a custom variable, \( p_{98} \), that evaluates to 1 when at least one signal is being routed to the first input of the Convolver effect and 0 otherwise, and a similar custom variable, \( p_{99} \), corresponding to the second input. (We use the symbols \( p_{98} \) and \( p_{99} \) because the music system
variables are denoted $p_1 \ldots p_{97}$ as indicated in Table 5.1.) Then by adding these custom variables to a rule group for ARL analysis, along with the Convolver effect variable, the following implies rule would be discovered:

$$\text{Convolver effect} = 1 \implies p_{98} = 1 \text{ AND } p_{99} = 1.$$  \hspace{1cm} (5.11)

This is equivalent to the generalised association rule expressed in 5.9 in Section 5.4.4, which could not have been discovered by ARL.

Custom variables can be used to capture other stylistically salient patterns identified in Section 5.3. For example, to ensure that performances using the EAI system begin with just a single track, the following steps could be followed:

1. Define a custom variable for each track that evaluates to 1 if a clip is playing in that track, and 0 otherwise. Ableton Live reports a negative value for tracks in which no clip is playing, so for example, a custom variable, $p_8$, indicating if the FM track is playing, would be given by:

$$p_8 = \begin{cases} 
1 & \text{if } p_1 > -1 \\
0 & \text{otherwise,}
\end{cases} \hspace{1cm} (5.12)$$

where as before, $p_8$ is used since the music system variables for the EAI system are denoted by $p_1 \ldots p_7$.

2. Having created 7 custom variables, $p_8 \ldots p_{14}$, one for each track, define a custom variable, $p_{15}$, the value of which indicates the number of tracks playing. That is:

$$p_{15} = \sum_{i=8}^{14} p_i. \hspace{1cm} (5.13)$$

3. Model the value of $p_{15}$ using a VMM. In this way, the temporal structure found in the number of tracks playing at a time, will be learnt from the training data set and reproduced in an agent’s performance. This is relevant to patterns 2 and 3, Table 5.4, both of which relate to the number of tracks playing in
performances with the EAI system.

Thus, the use of custom variables can increase the modelling capabilities of VMMs as well as association rules.

**Custom Variable Creation and Feature Selection**

There is a clear correspondence between the definition of custom variables and the feature selection activity of the standard machine learning workflow. Given the challenge that feature selection poses to users who are not machine learning experts, the adoption of this paradigm may seem problematic with respect to our aim that the ADTK be accessible to end users. However, we argue that it is necessary to introduce the feature selection activity to the design workflow, because it is the only way to allow a user to augment the model without introducing new machine learning algorithms. In addition, as alluded to in the chapter introduction, we envisage that the difficulty of the feature selection phase can be mitigated by presenting custom variables

1. as a preset selection that could be easily experimented with; and

2. in tandem with a set of ‘standard techniques’ for using them to capture common musical patterns.

In the following, we give a list of custom variable types that were selected to comprise the preset selection that would be available to the user. We justify the inclusion of each type with reference to the patterns identified in Section 5.3.

**Terminology and Notation**

Before listing the custom variable types we introduce some terminology and notation with reference to the two examples used above to introduce the paradigm of custom variables. The custom variables, \( p_{98} \) and \( p_{99} \), required for rule 5.11 are **ANY GREATER THAN** custom variables. This is because they evaluate to 1 if any of the variables
on which they depend are greater than a particular value, and they evaluate to 0, otherwise. The symbol for an ANY GREATER THAN custom variable is $\succ_{\text{ANY}}$. The value that is used to evaluate a custom variable is referred to as its critical value. In the case of $p_{98}$ and $p_{99}$, the critical value is 0. That is, $p_{98}$ evaluates to 1 if any of the variables on which it depends are greater than the critical value 0. The same is true for $p_{99}$. Not all custom variable types required critical values; see below.

In the case of $p_{98}$, the underlying variables (i.e. the variables on which $p_{98}$ depends), are those routing matrix variables that determine which signals are being routed to the first input of the Convolver effect. Those variables are $\{p_1, \ldots, p_{14}\}$ (see the details of the Montreal control parameters in table Table 5.1). We use the symbol $\mathcal{U}$ to denote the set of indices of the underlying variables, that is, for $p_{98}$, $\mathcal{U} = \{1, \ldots, 14\}$. The cardinality of $\mathcal{U}$, meaning the number of underlying variables, is denoted by $|\mathcal{U}|$ and in this case, $|\mathcal{U}| = 14$.

The custom variable used in the previous section to indicate the number of tracks playing, $p_{15}$, is a SUM custom variable. It has no critical value and its underlying variables are $p_8 \ldots p_{14}$. Thus the cardinality of the set of underlying variables, $|\mathcal{U}| = 7$. The symbol for a SUM custom variable is $\Sigma$. We note that this SUM custom variable, $p_{15}$, has other custom variables as its underlying variables. This is allowed in order that complex features can be created, however, circular dependencies are forbidden.

Finally, we introduce the following compact notation for defining custom variables. The expression to evaluate a custom variable is surrounded by square brackets. Inside the square brackets, a custom variable is defined using its symbol followed by a set of round brackets surrounding its critical value (if any) and a list of its underlying variables, all separated by commas. For example, the ANY GREATER THAN custom variable, $p_8$, used to indicate if the first track in the EAI system, $p_1$ is playing, is written:

$$p_8 = \left[ \succ_{\text{ANY}} (-1, p_1) \right]. \quad (5.14)$$
The SUM variable, $p_{15}$, can be written:

$$p_{15} = \left[ \Sigma (p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}) \right].$$  \hspace{1cm} (5.15)

(Note that the SUM custom variable has no critical value unlike the ANY GREATER THAN custom variable which has a critical value of -1 in the examples above.) However, this requires that the custom variables $p_8 \ldots p_{14}$ have already been defined. Instead, $p_{15}$ can be defined in a single expression;

$$p_{15} = \left[ \Sigma (>\ANY (-1, p_1), >\ANY (-1, p_2), >\ANY (-1, p_3), >\ANY (-1, p_4),
\hspace{1cm} >\ANY (-1, p_5), >\ANY (-1, p_6), >\ANY (-1, p_7)) \right]$$  \hspace{1cm} (5.16)

Thus, $p_{15}$ is a count of the number of variables with a value greater than -1. To finish, note that we do not envisage this notation being presented to a user of the ADTK, but it is useful for discussions of agent design in this thesis.

**List of Custom Variable Types**

A complete list of the 15 custom variable types is given in Table 5.8. Shown are the name of the custom variable type (column 1), its symbol (column 2) and the required cardinality of its set of underlying variables (column 3). In column 4, there is an entry if a critical value is required. In many cases, the critical value can be any integer ($c \in \mathbb{Z}$), however in some cases it must be a positive integer ($c > 1$). Finally, the fifth column of the table gives the formula to calculate the custom variable with index $x$ at time $j$, denoted $p_{x,j}$. We now give a brief description of the custom variable types before outlining how they may be used and combined to capture some of the stylistically salient patterns identified in Section 5.3.

The first 10 custom variable types are all straightforward comparisons that evaluate to 1 if true and 0 if not. There is some redundancy among the types. For example when used with a single underlying variable, the custom variable types ANY GREATER THAN and ALL GREATER THAN are equivalent, and this is also
### 5.4. A Rationale for the Learning Components of the ADTK

| Name               | Symbol | $|U|$ | Crit Val                      | Formula                                                                 |
|--------------------|--------|-----|-------------------------------|-------------------------------------------------------------------------|
| **Any Greater Than** | $\geq_{\text{ANY}}$ | $\geq 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 0; & p_{i,j} \leq c_i \ (i \in U) \\ 1; & \text{otherwise} \end{cases} \ |
| **All Greater Than** | $\geq_{\text{ALL}}$ | $\geq 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 1; & p_{i,j} > c_i \ (i \in U) \\ 0; & \text{otherwise} \end{cases} \ |
| **Any Less Than** | $<_{\text{ANY}}$ | $\geq 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 0; & p_{i,j} \geq c_i \ (i \in U) \\ 1; & \text{otherwise} \end{cases} \ |
| **All Less Than** | $<_{\text{ALL}}$ | $\geq 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 1; & p_{i,j} < c_i \ (i \in U) \\ 0; & \text{otherwise} \end{cases} \ |
| **Any Non-zero** | NZ$\text{ANY}$ | $\geq 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 0; & p_{i,j} = c_i \ (i \in U) \\ 1; & \text{otherwise} \end{cases} \ |
| **All Non-zero** | NZ$\text{ALL}$ | $\geq 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 1; & p_{i,j} \neq c_i \ (i \in U) \\ 0; & \text{otherwise} \end{cases} \ |
| **All Equal** | $=_{\text{EQUAL}}$ | $\geq 2$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 1; & p_{i,j} = p_{k,j} \ (i,k \in U) \\ 0; & \text{otherwise} \end{cases} \ |
| **Not All Equal** | $\neq_{\text{EQUAL}}$ | $\geq 2$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 0; & p_{i,j} = p_{k,j} \ (i,k \in U) \\ 1; & \text{otherwise} \end{cases} \ |
| **All Equal To** | $\equiv_{\text{EQUAL TO}}$ | $\geq 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 1; & p_{i,j} = c_i \ (i \in U) \\ 0; & \text{otherwise} \end{cases} \ |
| **Not All Equal To** | $\neq_{\equiv_{\text{EQUAL TO}}}$ | $\geq 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} 0; & p_{i,j} = c_i \ (i \in U) \\ 1; & \text{otherwise} \end{cases} \ |
| **Sum** | $\Sigma$ | $\geq 2$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ $\sum_{i \in U} p_{i,j}$ |
| **Previous** | PREV | $= 1$ | $c \in \mathbb{Z}$ | $p_{x,j} =$ \begin{cases} p_{i,j-1} \ (j > 1) \\ p_{i,j} \ (j = 1) \end{cases} \ |
| **Count** | COUNT | $= 0$ | $c > 1$ | $p_{x,j} =$ $j \mod c$ |
| **Combo** | COMBO | $\geq 2$ | $c \in \mathbb{Z}$ | | See Text |
| **Block** | BLOCK | $= 1$ | $c > 1$ | | See Text |

Table 5.8: The custom variable types available in the Agent Designer Toolkit at the time of writing. The symbol $p_{x,j}$ denotes the value of the custom variable with index $x$ at time $j$. 
true for ANY LESS THAN and ALL LESS THAN. In addition, certain types can be constructed from other types. For example, a custom variable equivalent to the ANY GREATER THAN type can be constructed using the ALL LESS THAN and ALL EQUAL TO types, that is:

\[
>_{\text{ANY}} (c, x, y, z) = [== (0, <_{\text{ALL}} (c + 1, x, y, z))],
\]

(5.17)

where \(c\) is the critical value of the ANY GREATER THAN custom variable, and \(x, y\) and \(z\) are its underlying variables.

The SUM custom variable type has been described above. It evaluates to the sum of the values of its underlying variables. Accordingly, it requires at least two underlying variables.

The PREVIOUS custom variable has a single underlying variable. It evaluates to the value of its underlying variable at the previous decision point. That is, at decision point \(j\), the PREVIOUS custom variable has the value taken by its underlying variable at decision point \(j - 1\). (At the first decision point, the value of the PREVIOUS custom variable is identical to that of its underlying variable.) The PREVIOUS custom variable was introduced for modelling patterns of changes across different variables, as in the example above.

The COUNT custom variable is unique in that its value is not related to that of any other variable; it has no underlying variable. Its value is simply \((j \mod c)\), where \(j\) is the index of the decision point (in the DNB and EAI systems, this is simply the bar number), \(c\) is the critical value associated with the COUNT custom variable, and \(\mod\) is the modulo operator. The COUNT custom variable is useful for learning patterns of hypermetrical structure in the training data set. For example, a COUNT custom variable with a critical value of 4 could be included in the association rule learning for the DNB system, to discover patterns related to the 4-bar hypermetrical structure (e.g. there is a fill on the fourth bar of every four-bar group; this is explained in more detail below).
The COMBO custom variable forms a tuple from the values of its underlying variables at each decision point. Each unique tuple formed is given an index and the value of the COMBO custom variable at a particular decision point is given by the index of the tuple that corresponds to the values of the underlying variables at that decision point. The COMBO custom variable is useful for modelling sets of underlying variables that have strict relationships which must be maintained; essentially, it enables a combination (combo) of variables to be treated as a single variable.

Finally, the BLOCK custom variable takes the sequence of values taken by its single underlying variable and splits it into blocks that are then treated as atomic units. The size of the blocks is specified by the critical value (an example is given below). Blocks may then be modelled using a VMM and during performance, a new value is drawn from the VMM every time a block ends, rather than at every decision point.

**Envisaged Uses of Custom Variables**

In the following, we give examples of how custom variables might be used to capture the stylistically salient patterns identified in Section 5.3 that could not be captured by VMMs or association rules.

**The State of an Ableton Live Track:** A custom variable can be created which evaluates to 1 if an Ableton Live track is playing and 0 otherwise. This can done by defining an ANY GREATER THAN custom variable with critical value -1 and the variable corresponding to the Ableton Live track in question as its single underlying variable. As previously mentioned, this works because Ableton Live reports a negative value for a track variable if no clip is playing, otherwise it reports the clip index (the first index is 0). For the Ableton Live track variable, \( p_x \), this would be written:

\[
p_{\text{isplaying}} = \left[ >_{\text{ANY}} (-1, p_x) \right].
\]  

(5.18)
The Number of Ableton Live Tracks Playing: Also as described previously, the SUM custom variable can be used to count the number of Ableton Live Tracks playing. For track variables, $p_1$, $p_2$, $p_3$, and $p_4$, this would be written:

$$ p_{num\text{playing}} = \left[ \sum_{\text{ANY}} (-1, p_1), >_{\text{ANY}} (-1, p_2), >_{\text{ANY}} (-1, p_3), >_{\text{ANY}} (-1, p_4) \right] $$

(5.19)

This is relevant to the stylistically salient patterns related to the number of tracks playing in the EAI system (see Table 5.4).

Describing the equivalence of certain variable values or clips in Ableton Live: As described in Section 5.3.3, the pattern underlying the sequence of drum clips in the Blue Drum track of the DNB system can be thought of as a sequence of four-bar segments in which each segment is one of AAAB, AABB or AAAA, where A is the basic rhythm and B is a randomly chosen fill (pattern 2, Table 5.6). While a VMM of 7th order or greater can capture this pattern, it is more restrictive than is necessary. For instance, the sequence AAAAAAAA will always be followed by the fill corresponding to the bright green colour in Figure 5.4, and this is not required to be consistent with the underlying style (any fill would be appropriate). It would be better to define a custom variable, $p_{\text{fill}}$ indicating if a clip containing a fill is being played, and then to model the temporal structure of $p_{\text{fill}}$ with a VMM. The value of $p_{\text{fill}}$ is given by:

$$ p_{\text{fill}} = \left[ >_{\text{ALL}} (0, p_1) \right] $$

(5.20)

This is because the Ableton Live clips containing fills are numbered from one upwards. This is illustrated in Table 5.9. This technique has the potential to alleviate the overfitting which can arise when a high-order VMM is trained on a small training data set.

Capturing ‘occasional synchronicity’: This is relevant to interaction between the BassGrain and Voice tracks of the EAI system (pattern 4, Table 5.4), and might be achieved using the COMBO custom variable type. Much of the time, the BassGrain and Voice tracks are are used independently, but there are periods when they are
5.4. A Rationale for the Learning Components of the ADTK

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Drum</td>
<td>( p_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rhythm / Fill</td>
<td>( p_{\text{fill}} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.9: Illustration of an abstract representation of the Blue Drum variable indicating whether the clip being played is the basic rhythm (0) or a fill (1). Shown are the values of the Blue Drum variable for the first 14 decision points of the example performance (see Figure 5.4).

turned on and off together. To model this, a custom variable, \( p_{\text{sync}} \) can be defined as:

\[
p_{\text{sync}} = \left[ \text{COMBO}(\geq \text{ANY} (-1, p_2), \geq \text{ANY} (-1, p_3)) \right]. \tag{5.21}
\]

That is, \( p_{\text{sync}} \) is 1 when the two tracks are either on together or off together, and 0 otherwise. Now the temporal structure of \( p_{\text{sync}} \) can be modelled using a VMM so that the two tracks combined activity is modelled separately from the specific clips being played.

**Capturing hypermetrical structure, method 1**: The four-bar hypermetrical structure of the Blue Drum track in the DNB system might be captured by creating two custom variables and adding them to a rule group for ARL analysis. Specifically, the following custom variables could be created:

\[
p_{8} = \left[ \Rightarrow \text{ALL} (0, p_1) \right] \tag{5.22}
\]
\[
p_{9} = \left[ \text{COUNT}(4) \right] \tag{5.23}
\]

where \( p_8 \) indicates if the Blue Drum track is playing the standard rhythm (i.e. clip 0), and \( p_9 \) is a COUNT variable with a critical value of 4. By adding these two custom variables to a rule group, the following rules would be found:

\[
p_{9} = 0 \implies p_{8} = 1 \tag{5.24}
\]
\[
p_{9} = 1 \implies p_{8} = 1 \tag{5.25}
\]

These correspond to the use of the standard rhythm in the first and second bar of
Parameter Decision Point (j)

<table>
<thead>
<tr>
<th>Label</th>
<th>Symbol</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>W.D. Filter</td>
<td>$p_{5}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W.D. Filter Block</td>
<td>$p_{\text{block}}$</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.10: Illustration of a Block custom variable. The W.D. Filter variable is treated in segments that are 4 decision points long.

Capturing hypermetrical structure, method 2: Along with the COUNT custom variable, we envisage that the BLOCK custom variable would be useful for learning hypermetrical structure in a training data set. As an example, consider the W.D. Filter variable in the DNB system. It has a rigid temporal structure in that its value is only changed at the beginning of a four-bar pattern as defined by the drum sequence (see pattern 3, Table 5.6). This pattern could be captured by a VMM. However, the order required would be 48 (this corresponds to the longest duration for which the parameter is left unchanged). This is not ideal since such a high order would cause a VMM-generated sequence to simply copy the sequence found in the training data set (overfitting would occur). Since the four-bar structure is stylistically important, but the precise sequence of four-bar segments is not, a better scheme might be to treat the W.D. Filter variable in segments of four bars and to sequence these segments with a low-order VMM. This could be done with a BLOCK custom variable, $p_{\text{block}}$. In this case, $p_{\text{block}}$ would have the domain $\{0, 1\}$ where 0 corresponds to the pattern $\{0, 0, 0, 0\}$ in the W.D. Filter variable and 1 corresponds to the pattern $\{1, 1, 1, 1\}$ (this is illustrated in Table 5.10).

Capturing the number of changes that take place: In certain music systems, that may include the EAI system, it may be important to ensure that variables do not change too quickly in succession. Thus, it may be important to model the temporal structure of a variable that represents the number of variables that change in value at each decision point. Such a variable can be created using an ALL EQUAL custom variable to test whether a PREVIOUS custom variable is equal in value to
its underlying variable. For example, to detect changes in a variable \( p_z \), a custom variable, \( p_{\text{change}} \), could be defined as follows:

\[
p_{\text{change}} = \left[ = (p_z, \text{PREV}(p_z)) \right]. \tag{5.26}
\]

This custom variable evaluates to 1 if the value of variable \( p_z \) is different from its value at the previous decision point, and 0 otherwise. Furthermore, it is possible to count the number of changes that occur at a particular decision point, by defining a custom variable, \( p_{\text{numchanges}} \), as:

\[
p_{\text{numchanges}} = \left[ \sum(p_{\text{change}1}, p_{\text{change}2}, \ldots, p_{\text{change}N}) \right]. \tag{5.27}
\]

The temporal structure of this variable could be modelled using a VMM.

**Summary**

We have introduced the paradigm of custom variables, whereby a model of musical performance can be augmented to capture patterns that cannot be captured by VMMs and association rules alone. We have proposed a strategy for reducing the conceptual barrier that we assume would be associated with the use of custom variables, due to their similarity to the feature selection phase of the traditional machine learning workflow. This strategy is characterised by the use of a preset selection of custom variables, and the provision of ‘standard’ techniques for achieving certain common musical patterns, which may include ones related to musical dynamics (e.g. counting the number of tracks playing) or hypermetrical structure (e.g. maintaining four-bar groupings). In Chapter 7 we report on a study of which one of the aims was to identify other common musical patterns and techniques for modelling them using custom variables. We note the method presented in Section 5.4.5 for arriving at a set of VMM-generated values that are consistent with all rules can incorporate custom variables by adding additional rules reflecting the relationships imposed by the custom variables. The implementation details of this are given in the next chapter.
In the following we outline how the class of models comprising VMMs, association rules and custom variables extends to interactive scenarios, before summarising the set of design activities that comprise using the ADTK to create a musical agent for arrangement-level decision making.

5.4.7 How the Model Works in an Interactive Context

So far we have only discussed the creation of models of musical decision making in a non-interactive context. Here, we briefly outline how the modelling procedures discussed so far would be adapted to an interactive context.

An interactive context is one in which a musical agent’s decision making should be influenced by the values of certain variables not under its control (i.e. arriving from the feature extractor). These variables might describe the actions of a musician, for instance, or they may describe the actions of a visitor to an interactive art installation, or even the output of other algorithmic processes. In any case, to design the way in which these variables should influence the agent’s musical decisions, they must be included in the training data set. Thus, in an interactive context, the training data set would include both input variables that are not controlled by the agent, and output variables that are controlled by the agent (these are the sets of variables denoted by \( p \) and \( q \), respectively, of the PQfe framework; see Section 1.3.1). For example, the training data set for the Montreal system could have been created by two musicians controlling the system cooperatively. An agent could be trained to replace one of the musicians by marking the variables under his control as output variables, and those under the control of the other musician as input variables.

To adapt the model to this interactive scenario, it is simply required that no input variables be modelled using VMMs (since the agent cannot control them) and that no input variables (or custom variables with input variables underlying them) appear in the consequents the association rules. By including input variables to be used in the evaluation of custom variables and in the antecedents of the association rules, dependencies can arise between the input variable values and the output variable
values, and thus the agent’s decision making will be influenced by the incoming data. During performance, then, the values of the input variables would be read just before an agent updates the values of the output variables at each decision point. In a number of the case studies reported in Chapter 8, interactive agents of the kind described here were developed.

### 5.4.8 The Learning Configuration

At this point, we have described the learning components underlying the ADTK and given a rationale for their design. We introduce the term learning configuration to refer to a complete specification of all the parameters of these components necessary to train a model of arrangement-level musical decision making from a training data set. A list of the elements of the learning configuration is given in Table 5.11.

It comprises the selection of custom variables (feature selection); the selection of groups of variables among which rules will be sought by the association rule learning algorithms; the configuration of the association rule learning algorithms; setting the orders of the VMMs; and finally giving each variable a priority relevant to executing the procedure described in Section 5.4.5 for using VMMs to generate values consistent with the rules.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature selection</td>
<td>Custom variables are created to help the machine learning algorithms find the salient musical patterns in the example data.</td>
</tr>
<tr>
<td>Rule group selection</td>
<td>Rule groups are defined, each of which contains a selection of variables and custom variables. The ARL algorithms only search for dependencies among the parameters within each group. Groups may have members in common.</td>
</tr>
<tr>
<td>ARL configuration</td>
<td>Parameters of the ARL algorithms are set. Each rule group has three learning parameters associated with it. They are (i) the minimum support, (ii) the minimum confidence, and (iii) the number of variables that can be involved in a rule.</td>
</tr>
<tr>
<td>VMM configuration</td>
<td>The maximum orders used by the VMMs modelling the variables.</td>
</tr>
<tr>
<td>Variable priority selection</td>
<td>A ‘priority’ is set for each VMM-modelled variable or custom variable.</td>
</tr>
</tbody>
</table>

Table 5.11: The activities related to feature selection and machine learning algorithm configuration (collectively: learning configuration) involved in designing a musical agent.
5.5 Overview of the ADTK Software

The ADTK software was developed in order to investigate the design of models of arrangement-level musical decision making using the learning components outlined in the previous section. A detailed walk-through of its interface can be found in Appendix A. In addition, the technical details of its implementation are given in the next chapter. An overview of the structure of the software, and how it fits into Max and Ableton Live, is given here.

The ADTK comprises two main software components that can operate independently of one another:

- The **Agent Designer** is a program for recording training data sets and using them to design musical agents. It has a standard point-and-click graphical-user interface and it outputs *agent files*, which store agent behaviours.

- The **Fast Performer** is a program that can read in an agent file and ‘run’ it, meaning that it can generate musical data in real-time according to the behaviour specified.

In the context of the *PQfe* framework (see Section 1.3.1), a combination of the
5.3. Overview of the ADTK Software

Fast Performer and an agent file forms a decision maker (see Figure 5.6). The Agent Designer is for producing the required agent file. In accordance with the overall goals of this research, the ADTK is general purpose, meaning that it is not specific to particular feature extractor or generator modules, however these components must be present.

Both the Agent Designer and the Fast Performer were implemented as plugins for Max. Max was chosen because it is a common platform for the development of systems for the performance of live electronic music, and also because of the ease with which it can be connected to other pieces of software or hardware using, for example, MIDI or Open Sound Control (a network-based music communication protocol for music [82]). A detailed walk-through of using the ADTK in Max is given in Appendix A.

An additional reason for implementing the Agent Designer and Fast Performer as Max plugins is that this made it straightforward to integrate them into Ableton Live using the Max for Live API, which is technology for embedding the Max platform in Ableton Live. Software written for Ableton Live in Max is referred to as a Max for Live Device, thus our package which includes the Agent Designer and Fast Performer for use in Ableton Live is the Agent Designer Device.

The Agent Designer Device was developed to provide a convenient way for users to experiment with creating interactive and generative music systems. As indicated in Section 2.1.1, Ableton Live allows musical material to be created using an interface that is accessible to users familiar with music production software, and in addition, it offers many ways for this material to be arranged and sequenced in a performance context using hardware controllers. Furthermore, the Agent Designer Device plugs directly into Ableton Live and can control many of Ableton’s parameters with minimal need for user configuration. This contrasts with using the ADTK in Max, where the user is required to supply the feature extractor and generator modules, as well as the infrastructure for communication with the decision maker. Again, a detailed walk-through of using the ADTK in Ableton Live can be found in
Figure 5.7: How Ableton Live combined with the ADTK fits into the PQfe framework.

Appendix A.

Figure 5.7 shows the combination of Ableton Live and the ADTK in the context of the PQfe framework (again, see Section 1.3.1). Ableton Live and the Agent Designer Device together provide both the feature extractor and the generator, as well as the mechanisms by which these components can communicate with the Fast Performer-based decision maker. Thus, an interactive or generative music system can be created entirely inside the Ableton Live environment without any requirement for conventional computer programming. However, while this provides a straightforward way to begin experimenting with the development of such systems, it is easily extensible: non-standard feature extractors and generators can easily be introduced using the Max for Live API.

5.6 Discussion

So far we have presented a design for a software toolkit for designing musical agents (i.e. generative models of arrangement-level musical decision making). We have not yet proved its efficacy as a means for developing agents either to emulate a
particular style. The toolkit will be characterised in these respect in Chapter 7, after giving details of its implementation in Chapter 6. In the following, we discuss the ADTK in relation to its conceptual forerunners and related work in the computer music literature.

5.6.1 Style Emulation

The ADTK is intended for developing agents to emulate the style in which an individual performs with a particular computer music system (as is required for programming by example). It can be seen in part as an extension of the ideas in the Continuator (see Section 2.3.3) to arrangement-level musical decision making involving parametric control of software instruments and effects. The Continuator was designed specifically for musical note data in two ways. First, as mentioned previously (Section 4.1.1), it uses reduction functions that are specific to musical notes and their perceptual characteristics. Second, its interactive mode, in which note generation is biased according to the musical context, requires a pre-programmed fitness function that describes the degree to which different note choices are compatible with other musical constraints, such as harmony or rhythm [136]. The ADTK does not require reduction functions, since VMMs are only applied to single variables, while in effect, the ARL-derived rules, together with the relationships imposed by the custom variables, form a binary fitness function that indicates whether or not VMM-drawn values are compatible with the current musical context (see the procedure described in Section 5.4.5). This part of the generation procedure is also related to the generate-and-test strategy used in Omax [72] to generate music according to a particular style.

In addition, we note that the stochastic sequencing of short musical patterns, as might be done by a single VMM controlling an Ableton Live track, is reminiscent of Mozart’s musical dice games [101]. These consisted of sequencing numbered musical fragments according to a constrained random process. Hodgson notes that though such a system ‘will not create new music by transforming a conceptual space
of possibilities, it will produce a huge number of variations’ [96, p. 61]. Of course, a model involving multiple variables with arbitrary stylistic dependencies between them is much more complex and we will argue in the following that from the outset, the ADTK showed promise of producing behaviours beyond simple variation of a demonstrated style.

### 5.6.2 Discovery of New Styles

Dean speculates on the promise of learning algorithms in the context of computer-generated improvisation and a number of parallels can be drawn between the characteristics of learning algorithms that give rise to his optimism, on the one hand, and on the other, the design possibilities made available by the ADTK. He gives a succinct outline of the most prevalent method for generating ‘pre-existing styles of composition and improvisation’:

‘Most cases ... have involved ... extracting salient examples of a style ... the computer algorithm then reads a specification of the context in which it is to operate ... and searches ... for the most appropriate musical fragment(s). Modest modification procedures are applied to the fragments, which are then sounded ... according to rules ... that reflect other aspects of the style,’ [67, p. 144].

While this was published in 2003, the general procedure (search for fragments to match the context, modify them and output subject to constraints) continues to see widespread use (the work in [136], [14] and [50] has been previously discussed).

Dean goes on to suppose that learning algorithms might remove some of the limitations of such systems, since they could potentially identify abstract and unexpected features by which to find matches in the database, and similarly abstract and unexpected dimensions along which to modify the fragments. He writes,

‘If the extraction of features (which could correspond to the “signatures” of which David Cope talks) can be relatively style neutral, then so can
their exploitation. The extent of modification ... could then be controlled, from random to consistent, permitting the production of music with any degree of relationship to the initial styles supplied. While musicians would immediately think of specifying the degree of variation along conventional musical parameters ... the program might well extract features couched in other terms ... this would allow the control of modification of material to act globally, or in quite different ways from these of musicians ... and to generate internal features are rules ...’ [67, p. 144].

The ADTK has similar potential for abstract matching and modification since it has direct counterparts to the features and signatures referred to here. The former term is used in the usual machine learning sense, and Dean speculates that algorithms might discover abstract, non-musical features and that these in turn might give rise to interesting results. Though in the ADTK such features will in general be introduced deliberately by a designer, they are available for use and they have similar potential for interesting results. As a simple example, it is possible in the ADTK to create a custom variable representing the number of music system parameter changes that occur at each decision point (see Section 5.4.6), and to model the temporal change of this abstract quantity with a VMM. Thus, an agent could be designed to produce parameter changes in similar number and temporal distribution to those found in the training data set, but completely different in the choices of the particular parameters that change and their values.

The second term mentioned above, signatures, is from Cope and is used to refer to recurring patterns in a corpus of musical works [64]. It is similar to our term, stylistically salient pattern. As discussed above, signatures might be modelled in the ADTK in a variety of ways, including through ARL analysis and temporal modelling of music system variables and custom variables. Moreover, by careful configuration, we envisage that they can be selected by the designer to allow for different degrees of variation.
Finally, Dean introduces the idea of a global variation controller to govern the extent to which the system would deviate from the demonstrated style, and

‘would certainly be a more diverse controller than any built into current MAX [sic] patches or other improvising patches’ [67, p. 145].

The ADTK allows the designer to set something akin to a global variation controller when creating a musical agent. At one extreme, using high-order VMMs and COMBO custom variables, for instance, it is possible to create an agent that simply reproduces examples from the training data set (overfitting). At the other extreme, an agent can be created that produces random parameter values (underfitting). This would give rise to renditions that depart radically from the training data set. Of course, it is in between these two extremes that lies the fertile ground made available by the ADTK, and its exploration is the focus of later chapters.

### 5.6.3 Models of Music Performance

Musical agents embody models of music performance and those produced using the ADTK are related to other models proposed in the literature. First, the idea of using constraints (i.e. rules and relationships between variables) to define a musical behaviour or style is not new. A constraint satisfaction problem (CSP) is a problem that is defined by a set of constraints [12] and in his oft-cited article on models for music improvisation, Pressing speculates that ‘Constraint satisfaction ... is a technique whose principles seem to apply to improvisation. The constraints are the referent, goals of the performer, stylistic norms, etc.’ [142]. More broadly, in the context of modelling music composition, rather than improvisation, constraint-based approaches have received considerable attention (see [9] for a review). However, constraint satisfaction problems in general are not suited to real-time applications—Pressing did not implement a constraint-based model—and this issue underlies much of the next chapter.

---

6Though, in the same way that white noise is infinite in its variety but perceptually monotonous, a completely random agent would not necessarily produce the maximum perceived variety.
Additionally, ADTK agents can be seen as examples of what Johnson-Laird refers to as neo-Lamarckian algorithms for creativity [100]. As opposed to neo-Darwinian algorithms which generate ‘ideas’ randomly and then filter them through some evaluation stage, a neo-Lamarckian algorithm uses a constrained generation stage to arrive at a set of ‘solutions’, one of which is then chosen randomly. In the ADTK, the constraints arise from the rules found by ARL analysis, the relationships between variables defined by the custom variables and the values of the input parameters that are not under the control of the agent. Once these constraints have been applied to the space of all combinations of music system variables, a random choice is made between ‘allowed’ sets of variable values (again, see next chapter for details).

### 5.6.4 Interactive Machine Learning

The ADTK is a PBE system and it supports a workflow that can be seen as a hybrid between those of standard machine learning and interactive machine learning (IML; both of these paradigms were introduced in Section 2.2). Such a workflow is depicted in Figure 5.8. It allows the user to iteratively augment and modify the training data set, but rather than compute a large number of features (as in IML), it retains feature selection as an important part of the design process through the creation of custom variables. In its support of this hybrid workflow, the ADTK owes much to Fiebrink’s *Wekinator* software [80], briefly discussed in Section 4.1.

There are, however, marked differences between the *Wekinator* and the ADTK. The *Wekinator* uses supervised machine learning algorithms exclusively and one of its primary envisaged applications is to ‘bridge the semantic gap’ (see [115] for more on this term) between sensed performer actions and musically meaningful interpretations of those actions [80, p. 28]. In these aspects, it is fundamentally different from the ADTK, since the ADTK uses only unsupervised machine learning algorithms to search for patterns in the performance data; and more importantly, while these algorithms may discover musically meaningful patterns if used naïvely, their potential for doing so is greatly increased by the user-performed feature se-
lection process: in large part it is the user that bridges the semantic gap and this is required because of the extreme paucity of training data for learning patterns in a very high-dimensional space\(^7\).

### 5.7 Conclusion

We have presented the Agent Designer Toolkit, a set of software modules for designing musical agents to perform with a given set of musical material, including parametrically controlled digital instruments and audio effects. Currently, the user interface gives direct access to the learning configuration (screenshots are shown in Appendix A). This is sufficient for the design of musical agents by non-programming end users, but in the future we envisage adding a more innovative interface to make the software more accessible and intuitive to use. In particular, the interface will incorporate standard techniques for modelling common musical patterns as well as preset learning configurations that can be simply selected. To inform the provision of such techniques and presets, we studied the software in practise both through a formal study of its modelling capabilities, through a set of case studies and through

\(^7\)This is not the space of parameter combinations but of compositions, i.e. the space of sequences of parameter combinations. Typical of the cases described in later chapters is the design of an agent based on four example performances. This amounts to the discovery of meaningful patterns from just four points in the space of compositions.
a usability study of the current version. These are reported in Chapters 7, 8 and 9, respectively. However, first, we describe the details of the implementation of the ADTK and in particular the *Fast Performer* module that performs musical decision making in real time.
Chapter 6

Implementation of a Real-Time Musical Decision Maker

6.1 Defining the CSP for an Agent ........................................... 178
   6.1.1 A Brief Overview of Constraint Satisfaction Problems .......... 178
   6.1.2 Representing an Agent’s Musical Decision as a CSP ............... 179

6.2 The Decision Making Procedure ......................................... 181

6.3 Solving the dynamic CSP in real time .................................. 185
   6.3.1 A Brief Outline of the CSP-solver implementation .................. 185
   6.3.2 Using BDDs to Solve CSPs ............................................ 186
   6.3.3 Implementation of the BDD-based Fast Performer ................. 188
   6.3.4 A Drawback of the Sugar Library ................................... 190

6.4 Performance of the BDD-based Fast Performer ....................... 192
   6.4.1 Results ........................................................................ 192

6.5 Discussion .......................................................................... 195
   6.5.1 BDD Performance ....................................................... 195
   6.5.2 Related Work in Computer Music ................................... 195
   6.5.3 Alternative Approaches .............................................. 196

6.6 Conclusion ........................................................................... 197
In this chapter, we describe the implementation of the mechanism that performs real-time musical decision making in the ADTK (see research question III-(ii), Section 1.6.2). Specifically, we outline how the rules discovered by the ARL analysis and the relations imposed by the custom variables together form a constraint satisfaction problem (CSP), which is a problem defined by a set of constraints between variables (more details are given below). To solve a CSP is to find a set of values for the variables such that all of the constraints are satisfied. During a musical performance, two steps must be taken at each decision point, (i) a CSP describing the current musical situation must be defined and then (ii) it must be solved. Furthermore, this must be carried out subject to real-time constraints, that is, with latency that is both small and strictly bounded.

An early implementation of the ADTK used a general purpose CSP solver software package to solve the CSP at each decision point. This was problematic because it was slow, frequently taking hundreds of milliseconds and sometimes much longer to arrive at a solution. As previously indicated, an important use of ADTK agents is to manipulate musical material in Ableton Live, typically updating parameter values once per bar and at 120 beats per minute (a quite modest tempo), a single 4-beat bar lasts for two seconds. Thus, there was little potential for exploring more complex agent designs or faster rates of decision making and a more efficient way of solving CSPs in real-time was required.

An additional drawback of general purpose CSP solvers is that they are based on backtracking search methods so it is usually not possible to predict the length of time it will take to find a solution. This means that during a performance, a hitherto unseen CSP could theoretically arise that takes an unusually long time to solve. Because of this potential issue, we did not seek a more efficient general purpose CSP solver on which to base our implementation of the real-time component of the ADTK. Instead, we implemented a new real-time decision maker that represents the
CSP corresponding to an agent’s decision making behaviour as a binary decision diagram (BDD). The BDD is a data structure that makes it possible to solve certain CSPs very efficiently and in a strictly bounded length of time. To our knowledge, this work was the first to use BDDs to efficiently solve musical CSPs.

This chapter is structured as follows. In the following section, we give an overview of CSPs and describe the construction of a CSP corresponding to an agent’s musical decision making task. Then in Section 6.2, we describe the decision making procedure that must take place at each decision point (the creation of the appropriate CSP and its solution). This description is independent of the method used to solve the CSP and we begin Section 6.3 with a brief outline of the CSP-based performer which is the early version of the real-time decision making component that was not based on BDDs. We then describe our more efficient implementation is the BDD-based real-time decision maker, which we refer to as the Fast Performer. This is followed in Section 6.4 with a characterisation of its performance in use with a selection of musical agents described in later chapters. Finally, in Section 6.5 we discuss the performance of the Fast Performer and its context in the computer music literature, and speculate about alternative implementations.

Before continuing, we note that this chapter draws on work first published in [123]. However, in that work an alternative version of the decision making procedure was implemented in which it was common for CSPs to arise which had many solutions. Thus, an important concern was to randomly choose a solution in order to maximise variety in the agent’s performance. The task of solving a CSP for a random solution is even more problematic for general purpose CSP solvers, since all solutions must be found and counted in order that one can be randomly chosen. If the number of solutions fluctuates greatly from one decision point to the next, this can be an additional cause of widely varying solution times (i.e. in addition to the unpredictability of the time required to solve the CSP for a single solution). For the alternative decision making procedure studied in [123], this provided another reason to use BDDs since they can generate random solutions very efficiently.
6.1 Defining the CSP for an Agent

6.1.1 A Brief Overview of Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is a problem defined by a set of variables, their domains and a set of constraints which are relations between the variables. To solve a CSP is to find a set of values that is consistent with all of the constraints. The topic of CSPs is dealt with comprehensively in a variety of texts (see, e.g. [12, 150]). Here we introduce the essential terminology required for the discussions in the remainder of this chapter.

A CSP with at least one solution is said to be feasible and one with no solutions is said to be infeasible. Constraints may be unary, involving only a single variable; binary, involving two variables; or tertiary involving more than two variables. Of particular note are global constraints, which are those capturing a ‘relation between a non-fixed number of variables’ [150, Ch. 7]. The most commonly cited example of a global constraint is ALLDIFFERENT, which specifies that the values of all variables involved in the constraint be unique⁴.

A variety of general purpose constraint solvers are available as standalone programs or software libraries. They differ primarily in the details of the algorithms used to find solutions and in the set of constraints that they support. Generally, solutions are found using backtracking search procedures (see, e.g. [12]). As mentioned in the introduction, regardless of their speed, this makes general purpose solvers unsuitable for real-time applications since it is generally not possible to predict in advance how long it will take to solve a CSP. However, the constraint-based representation is still a useful one since more efficient representations can be derived from it—such as the BDD alluded to in the chapter introduction—and in the following we give details of the CSP-based representation of an agent’s musical decision.

⁴There is an online catalogue of global constraints maintained by the CSP research community. It is available at: http://www.emn.fr/z-info/sdemasse/gccat/index.html.
6.1.2 Representing an Agent’s Musical Decision as a CSP

An agent created using the Agent Designer can be thought of as (i) a set of variables modelled using VMMs, and (ii) a set of constraints. At a particular decision point, the constraints arise from

1. The *implies* rules found by association rule learning;
2. The relations defined by the custom variables;
3. The values of the input variables (see Table 5.8);
4. The time index of the decision point (this sets the value of the COUNT custom variables);
5. The values from the previous decision point of variables that underly PREVIOUS custom variables; and finally
6. The values of variables drawn from VMMs.

In the following, we detail how each of these is represented using standard constraints (i.e. constraints universally supported by CSP solver software packages).

The *implies* rules discovered by association rule learning are standard constraints so they can be added directly to a CSP. In addition, most of the custom variable types also give rise to standard constraints. For example, GREATER THAN is a standard unary constraint and SUM is a well-known global constraint. Each PREVIOUS custom variable imposes an EQUALS constraint at each decision point, with its value set by that of its underlying variable at the previous decision point. Similarly, each COUNT custom variable imposes an EQUALS constraint, but in this case its value is determined by the index of the decision point (see Table 5.8). The COMBO and BLOCK custom variables each require sets of constraints to be added, as detailed below.

A COMBO custom variable has as its domain the indices of all unique combinations of values taken by its underlying variables in the training data set. The
relations it imposes are represented as additional *implies* constraints. For example, suppose $p_1$ and $p_2$ are two variables that underly a COMBO custom variable, $p_3$, and in the training data set, $p_1$ and $p_2$ take only three unique combinations of values: $(p_1, p_2) = \{(1,4), (1,5), (2,6)\}$. In this case, $p_3$ will have the domain $\{0,1,2\}$ and the following three *implies* rules will be added as constraints:

$$
p_3 = 0 \implies p_1 = 1 \text{ AND } p_2 = 4;
\quad p_3 = 1 \implies p_1 = 1 \text{ AND } p_2 = 5;
\quad p_3 = 2 \implies p_1 = 2 \text{ AND } p_2 = 6.
$$

The BLOCK custom variable is represented in a similar manner. The domain of a BLOCK custom variable comprises the indices of all unique ‘blocks’ of values taken by its underlying variable. Also, for each BLOCK custom variable, a COUNT custom variable is created with the same critical value (e.g. if the blocks are to have length 4, then the count variable will count from 0 to 3). Then a set of *implies* rules are created to express the relation between the BLOCK custom variable, its underlying variable, and the COUNT custom variable. For example, suppose a variable, $p_4$, underlies a BLOCK custom variable, $p_5$, with a critical value of 3, and in the training data set, $p_4$ takes only two different length-3 sequences of values: $\{(1,2,3), (4,5,6)\}$. In this case, a COUNT variable, $p_6$, will be created and the following *implies* rules will be added:

$$
\begin{align*}
&\begin{cases}
  p_5 = 0 \text{ AND } p_6 = 0 \implies p_4 = 1, \\
  p_5 = 0 \text{ AND } p_6 = 1 \implies p_4 = 2, \\
  p_5 = 0 \text{ AND } p_6 = 2 \implies p_4 = 3
\end{cases} \quad \text{Block 0} \\
&\begin{cases}
  p_5 = 1 \text{ AND } p_6 = 0 \implies p_4 = 4, \\
  p_5 = 1 \text{ AND } p_6 = 1 \implies p_4 = 5, \\
  p_5 = 1 \text{ AND } p_6 = 2 \implies p_4 = 6
\end{cases} \quad \text{Block 1.}
\end{align*}
$$
6.2 The Decision Making Procedure

Given the above method to represent a musical decision as a CSP, all that is required at each decision point is to

1. Create the CSP, and
2. Solve it.

In this section, we assume that there is an available tool to perform step 2 and describe a procedure for step 1. Much of the procedure is straightforward, involving the systematic addition of constraints. However efficiency gains can be made by taking into account which constraints change from one decision point to the next and which do not. In addition, care must be taken when adding values drawn from VMMs that the CSP does not become infeasible (see below).

We begin with the observation that the constraints arising from the association rules and the relations corresponding to custom variables (items 1-2 in the list of the previous section) do not change over the course of a performance (they are specific to an agent). We refer to the CSP composed only of these as the static CSP, \( \mathcal{S} \) (this symbol is a calligraphic ‘S’). At each decision point a dynamic CSP, \( \mathcal{D} \), can be formed by augmenting the static CSP with extra constraints arising from the values of the input variables; the PREVIOUS and COUNT custom variables; as well as new constraints corresponding to the values drawn from the VMMs (these are items 3-6 in the list of the previous section).

Algorithm listing 2 shows a procedure for choosing new values for the variables under the agent’s control by defining the dynamic CSP, \( \mathcal{D} \) and then solving it. The procedure is supplied with seven arguments:

- \( i \) - the decision point number, required for setting the values of COUNT and BLOCK custom variables;
- \( N \) - the maximum number of attempts that should be made to draw a value from each VMM that does not make the CSP infeasible (see below);
• $\mathcal{S}$ - the static CSP;

• $\mathcal{I}$ - a list of the input variables (i.e. those used to inform the decision making, usually describing the activity of the musician with whom the agent is interacting);

• $\mathcal{C}$ - a list of the COUNT custom variables;

• $\mathcal{P}$ - a list of the PREVIOUS custom variables; and

• $\mathcal{M}$ - a list of all variables modelled by VMMs sorted in order of priority, with the highest-priority variable first\(^2\). The only variables not included in $\mathcal{M}$ are those already mentioned (input, COUNT and PREVIOUS) and also those whose values are completely determined by higher-priority variables. There are many custom variable types for which this can be the case, (e.g. the value of an ANY GREATER THAN custom variable is completely determined if all of its underlying variables have higher priorities).

The procedure in Algorithm 2 begins by defining the dynamic CSP to be identical to the static one (line 2). Constraints are then incrementally added to the dynamic CSP so that it is first consistent with the input variable values, $\mathcal{I}$ (lines 3-6); then the COUNT variables, $\mathcal{C}$ (lines 7-11); and then the PREVIOUS variables, $\mathcal{P}$ (lines 12-16)\(^3\). The final phase of this incremental addition of constraints is to loop over the VMM-modelled variables, $\mathcal{M}$, in order of priority starting with the highest priority one (lines 17-30). In this phase, BLOCK variables are treated differently (lines 19-26) to non-BLOCK variables (lines 27-29) since the value of a BLOCK variable is only updated when a previous block has just ended. Apart from this difference, all VMM-modelled variables (BLOCK or not) are treated in the same way, using the 

CHOOSE AND SET VALUE procedure called on lines 22 and 28.

\(^2\)In the Agent Designer software, the user is not required to give a unique priority to each variable; two or more variables may be given equal priorities. Groups of variables with equal priorities are randomly reordered at each decision point.

\(^3\)Note that the rule learning algorithms in the Agent Designer are configured so that input, COUNT and PREVIOUS variables (and other variables that depend on them), cannot appear in the consequent of a rule. This means that constraining the values of these variables cannot lead to a CSP that is infeasible.
Algorithm 2 The decision making procedure

1: procedure UPDATEVALUES(i, N, S, I, C, P, M)  

2: \( \mathcal{D} \leftarrow \mathcal{S} \)  \( \triangleright \) Initialise the dynamic CSP identical to the static one

3: for all \( v \in I \) do  \( \triangleright \) For each input variable, \( v \)
4:  \( x \leftarrow \text{readFromInput}(v) \)  \( \triangleright \) Read its current value into \( x \)
5:  \( \mathcal{D} \leftarrow \text{addConstraint}(\mathcal{D}, v = x) \)  \( \triangleright \) Add constraint to \( \mathcal{D} \) such that \( v = x \)
6: end for

7: for all \( v \in C \) do  \( \triangleright \) For each COUNT variable, \( v \)
8:  \( c \leftarrow \text{getCriticalValue}(v) \)  \( \triangleright \) Get its critical value
9:  \( x \leftarrow i \mod c \)  \( \triangleright \) Calculate its value
10:  \( \mathcal{D} \leftarrow \text{addConstraint}(\mathcal{D}, v = x) \)  \( \triangleright \) Add the corresponding constraint
11: end for

12: for all \( v \in P \) do  \( \triangleright \) For each PREVIOUS variable, \( v \)
13:  \( v' \leftarrow \text{getUnderlying}(v) \)  \( \triangleright \) Get its underlying variable, \( v' \)
14:  \( x \leftarrow \text{getPreviousValue}(v') \)  \( \triangleright \) Get the previous value of \( v' \)
15:  \( \mathcal{D} \leftarrow \text{addConstraint}(\mathcal{D}, v = x) \)  \( \triangleright \) Add the corresponding constraint
16: end for

17: for \( j \leftarrow 1, |M| \) do  \( \triangleright \) Iterate over remaining variables in priority order
18:  \( v \leftarrow M(j) \)  \( \triangleright \) \( v \) is the current variable being processed
19:  if \( v \) is a BLOCK custom variable then  \( \triangleright \) If \( v \) is a BLOCK variable
20:     \( c \leftarrow \text{getCriticalValue}(v) \)  \( \triangleright \) Get its block size
21:     if \( i \mod c = 0 \) then  \( \triangleright \) If we are at the start of a new block
22:       \( \text{chooseAndSetValue}(v, \mathcal{D}, N) \)  \( \triangleright \) Choose a new value (see text)
23:     else  \( \triangleright \) Else we are not starting a new block
24:       \( x \leftarrow \text{getPreviousValue}(v) \)  \( \triangleright \) So get the previous value of \( v \)
25:       \( \mathcal{D} \leftarrow \text{addConstraint}(\mathcal{D}, v = x) \)  \( \triangleright \) And set \( v \) equal to that value again
26:     end if
27:  else  \( \triangleright \) Else \( v \) is not a BLOCK variable
28:     \( \text{chooseAndSetValue}(v, \mathcal{D}, N) \)  \( \triangleright \) Choose a new value (see text)
29:  end if
30: end for

31: return getRandomSolution(\( \mathcal{D} \))  \( \triangleright \) Solve the dynamic CSP for a random solution

32: end procedure
The CHOOSEANDSETVALUE procedure by which a value is drawn for each VMM-modelled variable is shown in Algorithm listing 3. It can be explained as follows. Ideally, for each VMM-modelled variable, \( v \), a value, \( x \), would be drawn from its VMM and the corresponding \textsc{Equals} constraint, \( v = x \), would be added to \( D \). However, since values are drawn independently from the VMMs, it is possible that adding such a constraint would make \( D \) infeasible. For example, values might be drawn that are not consistent with the ARL-discovered rules. To avoid this eventuality, a number of attempts, \( N \), are made to draw a value from the VMM that does not make \( D \) infeasible (if a value does make the CSP infeasible, it is rejected).

### Algorithm 3

The procedure to choose a value for a VMM-modelled variable

1: procedure CHOOSEANDSETVALUE(\( v, D, N \))

2: \( k \leftarrow 1 \) \quad \text{\( \triangleright \) Initialise the number of attempts to 1}

3: \( B \leftarrow \emptyset \) \quad \text{\( \triangleright \) Initialise the set of ‘bad’ values to the empty set}

4: repeat \quad \text{\( \triangleright \) Try up to \( N \) times to draw a value that leads to a feasible CSP}

5: \( x \leftarrow \text{drawMarkovValue}(v, B) \) \quad \text{\( \triangleright \) Draw a value from the VMM for \( v \) (see text)}

6: \( \mathcal{T} \leftarrow \text{addConstraint}(D, v = x) \) \quad \text{\( \triangleright \) Create new CSP that includes \( v = x \)}

7: \( k \leftarrow k + 1 \) \quad \text{\( \triangleright \) Increment the number of attempts}

8: \( B \leftarrow B \cup x \) \quad \text{\( \triangleright \) Add \( x \) to \( B \) (only used if the new CSP is infeasible)}

9: until isFeasible(\( \mathcal{T} \)) or \( k > N \) \quad \text{\( \triangleright \) Check if another attempt is necessary and available}

10: if isFeasible(\( \mathcal{T} \)) then \quad \text{\( \triangleright \) If we have found a suitable value for \( v \)}

11: \( D \leftarrow \mathcal{T} \) \quad \text{\( \triangleright \) Then we can keep the new CSP with the added constraint}

12: end if \quad \text{\( \triangleright \) Otherwise the procedure ends with \( D \) unchanged}

13: end procedure

To improve the chance of drawing a suitable value within \( N \) attempts, the procedure used to draw values from a VMM is a modified version of that presented in Section 2.3.2. Each time it is called, the procedure, \textsc{drawMarkovValue}, is supplied with a set of ‘bad’ values (i.e. ones that have been found in previous attempts to make the CSP infeasible). This set is denoted \( B \) in the algorithm listing. As in the procedure presented in Section 2.3.2, \textsc{drawMarkovValue} descends the VMM tree as far as possible (limited either by the tree itself or the maximum order of the VMM). However, before randomly selecting a continuation, it removes all of the ‘bad’ values from the list of possible continuations. If this results in an empty list of
continuations, then the order of the VMM is reduced as necessary until a non-empty list of continuations is found from which to draw a value. It is guaranteed that such a list will be found since the continuations list associated with the root node of the VMM tree must contain all of the values in the domain of the variable. If after $N$ attempts a value cannot be found, then the VMM for that variable is ignored for this decision point. Note that if the size of the variable’s domain is less than or equal to $N+1$, this procedure is guaranteed to find a suitable value (i.e. the VMM will not be ignored).

Returning to Algorithm 2, the decision making procedure, we note that taking the VMM-modelled variables in priority order ensures that those with high priorities are less likely to have their VMM ignored than those with low priorities, and that with the highest priority is guaranteed not to have its VMM ignored. Once CHOOSEAND-SETVALUE has been called for each VMM-modelled variable, the dynamic CSP is completely specified for that decision point. If a suitable value was found for every variable, then the CSP will have only a single solution. However, if VMMs were ignored for some variables then the CSP may have more than one solution in which case, ideally, a random solution would be chosen to maximise variety in the agent’s decision making. Thus, line 31 Algorithm 2, the final step in the decision making procedure is to return a random solution to the CSP, rather than the first solution that is found.

6.3 Solving the dynamic CSP in real time

6.3.1 A Brief Outline of the CSP-solver implementation

The procedure outlined above is a straightforward way of finding a set of values for the music system variables and custom variables that is consistent with all of the ARL-derived rules and custom variable definitions, while also using as many VMM-sampled values as possible (with respect to the user-defined priorities). However, as described above, its implementation is non-trivial since the dynamic CSP may
take too long to solve, and furthermore, this may not be known until performance
time: Since it is generated on-the-fly at each decision point, it is always possible that
a dynamic CSP will arise that is hitherto unseen and insoluble in a short enough
time. Despite this problem, which arises from the characteristics of general purpose
CSP solvers mentioned in Section 6.1.1, early development versions of the ADTK
did use a general purpose CSP solver to implement Algorithm 2. We began this
way both to reduce development overhead; and to allow experimentation with
different constraints and custom variables. The latter was pertinent because general
purpose constraint solvers usually support a wider range of constraints than solvers
embodies other approaches to solving CSPs.

The performer module that preceded the Fast Performer was implemented
in Java using a constraint solver library called Choco\(^4\). At each decision point,
the dynamic CSP was created before solving it for all solutions and choosing one
randomly. We found that this implementation generally worked well with agents
designed to control a small number of variables, or a set of variables with small
domains. However, as indicated above, it was too slow for use with more complex
agent designs.

Next, we describe an alternative way to implement Algorithm 2 above. It involves
representing the CSP as a binary decision diagram (BDD). As mentioned in the
chapter introduction, this data structure makes it possible to find random solutions
to CSPs in a completely predictable length of time, and usually with great efficiency.

### 6.3.2 Using BDDs to Solve CSPs

Binary decision diagrams (BDDs) are representations of Boolean functions (i.e. func-
tions of Boolean variables, see, e.g. [106]) and in this section, we show how they
can be used for real-time musical decision making. To make use of a BDD, we
first transform our CSP into a Boolean satisfiability problem (referred to as a SAT
problem). This can be understood as a CSP in which all of the variables are Boolean;

\(^4\)Version 2.1 was used, available at: [www.emn.fr/z-info/choco-solver](http://www.emn.fr/z-info/choco-solver)
they can only take on values of true or false (see, e.g. [84] for a thorough overview). A SAT problem is equivalent to a Boolean function that gives an output of true if all the constraints are satisfied and false otherwise.

Before continuing, and with reference to our reasons for using a general purpose constraint solver in the prototype CSP-based performer (see Section 6.3.1), we note that not all constraints supported by a general purpose constraint solver can be readily translated to SAT. However, at the time of implementing the BDD-based Fast Performer, the set of custom variables had been defined so it was known that the required constraints could be translated to SAT by available tools.

Once the Boolean function representing the static CSP has been created, it can be transformed into a BDD, which represents the Boolean function as a directed-acyclic graph [37]. This two-step process to compute the BDD (CSP \(\rightarrow\) SAT, SAT \(\rightarrow\) BDD), can be performed offline (i.e. at design time). Once it has been completed, it is possible to perform Algorithm 2 in a predictable length of time, and usually with great efficiency. This is because the BDD has special properties which make it a very attractive representation in the context of real-time musical decision making. Using a BDD, the following operations are possible:

- **Check if there are any solutions:** This is required for the isFeasible routine called in the CHOOSEANDSETVALUE procedure. Using a BDD, it can be performed in constant time [106].

- **Conjunction:** This is the operation required to add a constraint to a BDD; it is conjoined with a second BDD in which the new constraint is true. Theoretically, it can be carried out in \(O(B_1B_2)\) time where \(B_1\) is the number of nodes in the first BDD and \(B_2\) is the number of nodes in the second [106]. However, practical implementations do not generally achieve this in full due to the implementation-specific intricacies of the way computer memory is used and allocated [162]. This operation is critical to the implementation of Algorithm 2, since it is required each time a variable is set to a particular value (i.e. the
ADDCONSTRAINT procedure). At each decision point, values are set for the input variables; the COUNT and PREVIOUS custom variables; as well as when values are chosen for the VMM-modelled variables.

- **Count the number of solutions**: This is required for choosing a random solution. It can be done in $O(nB)$ time, where $n$ is the number of variables in the Boolean function and $B$ is the number of nodes in the BDD [106].

- **Choose solutions randomly**: Once the number of solutions has been counted, a random solution can be selected, with each solution being equally likely. This can be performed in $O(n)$ time or less [106].

All of these operations require the BDD to be constructed to begin with. While the process of transforming a Boolean function into a BDD can be done offline, and therefore is not time-critical, it has two pitfalls which we mention here. First, for a given Boolean function, the size of the BDD (i.e. the number of nodes, $B$) is very sensitive to the way in which the Boolean variables are ordered (i.e. the order in which they are encountered as the decision diagram is traversed, again see [37]). A sub-optimal variable ordering can lead to a BDD which is much greater in size than that which would result from the optimal variable ordering. The problem of finding the optimal variable ordering is very hard to solve (it is coNP-complete [106]). However, heuristic algorithms exist which can usually find good variable orderings. The second pitfall is that some Boolean functions simply cannot be compactly represented using a BDD, even if the optimal variable ordering is known [106]. However, these problems have not prevented BDDs from being successfully used in many different problem domains. In the next section, we give details of the software implementation of the BDD-based real-time decision making procedure.

### 6.3.3 Implementation of the BDD-based Fast Performer

As described in the previous chapter, the Agent Designer Toolkit is implemented as a pair of plugins for Max. The Agent Designer plugin is for recording training
data and designing agents. It is written in Java and has its own GUI. The Fast Performer is a BDD-based real-time decision maker. Its core functionality is to compute Algorithm 2 using a BDD to represent the CSP. It was implemented in C++. In this section, we present those aspects of the implementation of the Agent Designer and Fast Performer, that are related to the creation and solution of CSPs in real time.

The construction of the CSP from the rules learned by association rule learning and the relations determined by the custom variables, is performed by the Agent Designer (see Figure 6.1). Then the conversion from CSP to BDD is performed as a two-stage process as outlined in Section 6.3.2. The first stage of this process (CSP → SAT) is performed by the Agent Designer, using the Java-based Sugar CSP library\(^5\) [167]. The SAT representation, along with the VMMs and other information, is loaded into the Fast Performer.

The Fast Performer is based on the CUDD package\(^6\) [163] for creating and manipulating BDDs (as well as other types of decision diagrams). This is used to convert the SAT representation into a BDD at load-time, and to perform the BDD projection operations and find random solutions at performance time\(^7\). Though the conversion

---

\(^5\)See: bach.istc.kobe-u.ac.jp/sugar

\(^6\)See: vlsi.colorado.edu/~fabio/CUDD

\(^7\)The particular heuristic variable ordering scheme used is the CUDD implementation of Rudell’s sifting algorithm [155]. This was found to give adequate and consistent performance across a variety of agent designs.
from SAT to BDD can in theory take a long time—and so might better be done as a separate, offline process so as not to delay the loading of an agent—we have not found it to take very long in practise (see Section 6.4).

### 6.3.4 A Drawback of the Sugar Library

The *Sugar* library was chosen for its efficiency and ease of use. However, the method used by Sugar to encode integer-valued variables as binary variables has an effect on the distribution from which random solutions to a CSP are drawn. While a BDD can be used to draw a random solution to the SAT problem with all solutions being equally likely, the conversion of the SAT solution to its integer counterpart does not maintain this uniform distribution. This is a minor issue since frequently the CSP that is created at a decision point has only one solution so it is unnecessary to solve for a random solution. Nevertheless we give details here.

As opposed to the standard method for encoding integer variables in a binary representation, Sugar uses the order-encoding scheme [167]. Using this encoding, the comparison $p_x \leq a$ is encoded by a binary variable for $\{l(x) \leq a < u(x)\}$, where $p_x$ is the variable to be encoded; and $l(x)$ and $u(x)$ are the lower and upper bounds, respectively, of its domain [168]. Thus, a variable with domain size $D$ is encoded using $D - 1$ binary variables. For example, an integer variable with domain $\{1, 2, 3, 4\}$ (i.e. $D = 4$) and having the value 2 would be encoded as $\{0, 1, 1\}$. As implemented in Sugar, the decoding scheme uses the first 1 in the binary sequence to determine the value of the integer variable. Thus, returning to the example, the binary encodings $\{1, 1, 1\}$, $\{1, 0, 1\}$, $\{1, 1, 0\}$ and $\{1, 0, 0\}$ are all decoded as the integer value 1. This means that if the binary variables are chosen randomly with 0 and 1 having equal probabilities, then the probability of each value in the domain

---

8At the time of writing, Sugar, in combination with a SAT solver, has performed well in a number of recent International CSP Solver competitions; see [http://bach.iste.kobe-u.ac.jp/sugar/csc.html](http://bach.iste.kobe-u.ac.jp/sugar/csc.html).
being chosen, is given by

\[
P[p_x = l(x)] = \frac{1}{2},
\]

\[
P[p_x = l(x) + 1] = \frac{1}{4},
\]

\[
P[p_x = l(x) + 2] = \frac{1}{8},
\]

\[\vdots\]

\[
P[p_x = u(x) - 1] = \frac{1}{2^{(D-1)}},
\]

\[
P[p_x = u(x)] = \frac{1}{2^{(D-1)}}.
\]

Where the music system variables themselves are binary, this does not represent a problem since it corresponds to a uniform probability distribution over the domain \((P[p_x = l(x) = 0] = P[p_x = u(x) = 1] = \frac{1}{2})\). However, the probabilities decrease exponentially so for variables with large domains, the probability of a high value being drawn becomes vanishingly small.

In the Fast Performer, the \texttt{CHOOSEANDSETVALUE} procedure can only fail to find a suitable value for a variable if the domain of the variable is greater than \(N + 1\), where \(N\) is the number of attempts made (see Section 6.2). In practise, we typically set \(N\) to 10 or more, so this means that only variables with quite large domains (greater than \(N + 1\)) may have their VMM ignored. Such variables have not frequently arisen so we do not regard this as an important issue. When CSPs with multiple solutions do arise, the solution will be drawn from a non-uniform probability distribution over the space of possible solutions, and this will affect the agent’s decision making behaviour. However, we envisage that in cases where this results in undesirable decision making, it will be possible to address the issue through changes to the agent’s design.
6.4 Performance of the BDD-based Fast Performer

Our motivations for basing the Fast Performer on BDDs were first, that BDDs provide an efficient way of solving certain CSPs and second, that BDD theory provides strict bounds on the time it will take to arrive at a solution (once the BDD has been created). In this section, we empirically show that the Fast Performer fulfils our requirements by characterising its performance with a variety of different musical agents. The agents used for the measurements reported here were taken from the study in the following chapter in which participants were asked to judge the musical performances of a range of agents. Each participant provided musical material and example performances, and judged six different agents specifically designed to generate new performances with the same material (details are given in the next chapter). Thus, there were 48 agents created in total and six were judged by each participant. In the study reported here, we characterise the performance of the Fast Performer for each participant’s winning agent, that is, the one that he/she judged to be best overall. For each agent, repeated measurements were made of the time taken to update the values of the variables under its control. The mean and standard deviation were calculated from 1000 update measurements. In addition, for the BDD associated with each agent, the number of Boolean variables, the number of nodes and the time taken to create the BDD were also recorded. The latter was averaged over 10 trials.

6.4.1 Results

Update Times

For each of the seven ‘winning’ agents, a histogram showing the distribution of times taken to perform an update is shown in Figure 6.2. The agents are labelled P1-P7, according to the notation used to refer to the seven participants in the following chapter. Note that in the figure, the ranges shown on the horizontal and vertical axis differ throughout. For four of the seven agents the maximum time taken for any
6.4. Performance of the BDD-based Fast Performer

The variations in update times between agents may be due to a number of factors. Most importantly, different agents require a different number of conjunction operations (see previous section) to be performed at each update. A conjunction operation is required for each variable that has to be constrained to a particular value, and the set of such variables comprises the input variables, the COUNT custom variables, the PREVIOUS custom variables and the VMM-modelled variables. The time taken to perform a particular conjunction operation depends on the number of nodes in the two BDDs being conjoined and these sizes in turn depend on a variety of factors including the constraints that have already been added, and the size of the domain of the variable being constrained.

Table 6.1 shows details of each BDDs, including the number of nodes, the number of variables requiring conjunction operations and the product of these two quantities, \( P \). The three agents requiring the longest update times (P1, P3, P6) all featured the highest values of \( P \). The values of \( P \) do not entirely account for the update times.
Chapter 6. Implementation of a Real-Time Musical Decision Maker

Table 6.1: Fast Performer update times for a variety of different agents. From left to right, the columns contain the agent label; the mean time per update; the standard deviation of the time taken per update; the number of Boolean variables in the BDD; the number of nodes in the BDD; the time taken to create the BDD; the total number of variables requiring conjunction operations, $N_c$, made up of input, Count, Previous and VMM-modelled variables (see text); and finally the product, $P$, of $N_c$ and the number of nodes in the BDD.

<table>
<thead>
<tr>
<th></th>
<th>$\mu$ (ms)</th>
<th>$\sigma$ (ms)</th>
<th>$#$ Bool Vars</th>
<th>$#$ Nodes</th>
<th>Creation Time (s)</th>
<th>$N_c$</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>11.34</td>
<td>3.51</td>
<td>262</td>
<td>127353</td>
<td>54.32</td>
<td>14</td>
<td>1,782,942</td>
</tr>
<tr>
<td>P2</td>
<td>0.77</td>
<td>0.36</td>
<td>193</td>
<td>5089</td>
<td>1.01</td>
<td>32</td>
<td>162,848</td>
</tr>
<tr>
<td>P3</td>
<td>30.49</td>
<td>10.47</td>
<td>342</td>
<td>42609</td>
<td>19.64</td>
<td>28</td>
<td>1,192,052</td>
</tr>
<tr>
<td>P4</td>
<td>0.03</td>
<td>0.01</td>
<td>40</td>
<td>467</td>
<td>0.02</td>
<td>8</td>
<td>3,736</td>
</tr>
<tr>
<td>P5</td>
<td>5.50</td>
<td>1.68</td>
<td>251</td>
<td>19617</td>
<td>50.62</td>
<td>21</td>
<td>411,957</td>
</tr>
<tr>
<td>P6</td>
<td>50.12</td>
<td>1.44</td>
<td>347</td>
<td>21860</td>
<td>15.67</td>
<td>52</td>
<td>1,136,720</td>
</tr>
<tr>
<td>P7</td>
<td>4.55</td>
<td>1.36</td>
<td>165</td>
<td>29563</td>
<td>4.05</td>
<td>19</td>
<td>561,697</td>
</tr>
</tbody>
</table>

taken, however, the other factors mentioned above also play a role.

A number of factors may account for the variation in update times for any particular agent. These include the values to which variables are being constrained (which affects the conjunction operation). In addition, the CUDD library, used to perform the BDD calculations, opaquely performs various operations related to memory management such as garbage collection. This may account for the broad, perhaps bimodal, distribution apparent in the update times for agent P3 in Figure 6.2.

**BDD Creation Times**

The time taken to create each BDD is also shown in Table 6.1. The longest times are between approximately 50 s and 55 s, for agents P1 and P5. Clearly, the BDD creation times vary quite widely. Informally, we have observed that global constraints involving large numbers of variables (for example, the Sum of the tracks playing) can occasionally give rise to somewhat longer BDD creation times (though not necessarily large BDDs). Agent P1 involves a sum over eight tracks and agent P5, a sum over eleven tracks. However, this does not always arise.
6.5 Discussion

6.5.1 BDD Performance

We have not concretely defined an upper limit on the allowable update time. Our aim when implementing the Fast Performer was that it be as fast as possible in order to maximise the variety of agents that could be designed and the variety of contexts in which they could be studied. A notional upper limit on the allowable update time for an agent could be taken as the duration of one crotchet at 220 beats per minute (approximately 272 ms). In this case, the most computationally expensive agents studied above require a maximum update time smaller than this by a factor of more than 4 (calculated using the largest observed update time observed which was approximately 60 ms).

In addition, these results are very broadly consistent with BDD theory, which says that the update time required by the Fast Performer module only depends on the number of variables and the BDD sizes, the latter being governed by the rules and custom variables and by the sizes of the domains of the variables (as mentioned above, these affect the conjunction operation). However, as noted in Section 6.3.2 practical implementations of BDD manipulation algorithms do not typically achieve the theoretical performance, and the spread in update times can be attributed to this. (Note that the BDD operations take far more CPU time than, for instance, drawing values from VMMs.) It may be possible to reduce the variation in update times by managing the details of the memory management performed by the CUDD library.

6.5.2 Related Work in Computer Music

There has been some previous research into the use of constraints in real-time, interactive music applications [8, 138], though only with manually specified constraints on musical data. As previously mentioned, it is impossible to know in advance, how long a constraint solver will take to find a solution. Thus, in [8] where a constraint solver is used in a real-time scenario, the workaround is to use a timeout which stops
the search if a solution cannot be found within a specified duration. In contrast, the predictability of our BDD-based solver is a great advantage in real-time applications.

To our knowledge, this is the first application of BDDs to solving musical constraint problems in real time. In contemporary work, BDDs have been used to implement an efficient relational constraint solver for music [174]. However, this was in the context of modelling musical composition, rather than real-time performance.

6.5.3 Alternative Approaches

In our use of the Sugar Library for converting CSPs to SAT problems, our implementation prioritises efficiency and predictability in favour of true uniform sampling from the solution space. It is the use of order encoding in Sugar that distorts the probability distribution over the space of solutions. There are other methods for encoding integer variables as binary variables, such as direct encoding [178], that do not have this property. However they are less efficient [169] and to our knowledge at the time of writing, there are no off-the-shelf tools for performing the CSP $\rightarrow$ SAT conversion using other encodings.

Apart from BDDs, there do exist general purpose solvers for solving Boolean satisfiability problems, known as SAT solvers. In theory, they have the same disadvantages as general purpose constraint solvers (i.e. they lack strict bounds on the solution time). However, modern SAT solvers are extremely efficient, even with very large systems of variables and constraints [87]. In addition, there are methods for drawing random solutions to SAT problems without having to enumerate the entire solution space, such as that described in [86]. If a class of musical agents were found for which BDDs were unsuitable, then SAT solvers may provide a practical alternative.
Constraint-based systems are a powerful way for musicians to make musical decision making agents. Constraints provide a way of formally specifying a set of relations between musical elements that can accurately capture a musician’s conceptualization of their music. However, CSPs are not generative models, meaning that although they provide an accurate representation of a set of musical constraints, they cannot be used to derive musical decisions. In addition, they can be difficult to solve in a real-time context. The BDD provides a real-time generative representation of a CSP in a way that is practically useable by musicians.

In our view, the Fast Performer provides a useful and practical solution to the problem of making musical decisions according to rules and constraints in real time. Thus, it constitutes a good solution to the problem outlined in research question III-(ii) (see Section 1.6.2). It is imperfect in that randomly chosen solutions are not equally likely, but this is an idealised aim and it is yet to be seen if there are real ramifications for the ADTK as a tool for designing the behaviour of musical agents. We will address this question, as well as others previously raised in the coming chapters which focus on the use of the ADTK in practise.
Chapter 7

Modelling Arrangement-Level Musical Decision Making with the ADTK

7.1 Methods ................................................................. 201
7.1.1 Participants .......................................................... 202
7.1.2 Recording and Post-Processing Example Performances ........................................ 202
7.1.3 Agents ................................................................. 206
7.1.4 Agent Performances ................................................ 212
7.1.5 Feedback Questionnaire .......................................... 213

7.2 Results: Participant Submissions and Responses ................................................... 218
7.2.1 Compositions and Examples Submitted .......................................................... 218
7.2.2 Participant Responses .................................................................................. 230
7.2.3 Direct Assessment of Agent ’Usability’ ....................................................... 249
7.2.4 Uses of ADTK Agents ............................................................................. 251

7.3 Results: Agent Design Techniques ................................................................. 254

7.4 Discussion ........................................................................... 260
7.4.1 The Modelling Capabilities of ADTK Agents .............................................. 260
7.4.2 Informing the Design of a Higher-Level User Interface .................................... 264
7.4.3 Possible Sources of Error ................................................................. 269

7.5 Conclusion .......................................................................... 271
At this point, we have described the Agent Designer Toolkit (ADTK) and its implementation. In addition, we have identified research questions relating to the range of musical performance behaviours that can be captured with ADTK agents, and how the design of a high-level user interface might be carried out in order that the software not require users to develop their own techniques for using custom variables and other features (see Section 5.1). In this chapter, we report on a study conducted with the aims of

- characterising the modelling capabilities of the software by investigating the extent to which an ‘expert’ user of the software can design musical agents suitable for a live performance context (this is the primary objective and it corresponds to research question III-(iii), Section 1.6.2);

- characterising the usefulness of a variety of pre-determined learning configurations that might be made available as options in a high-level user interface for the Agent Designer (and thus, removing the need for a user to create learning configurations of their own);

- identifying techniques for using the Agent Designer to capture specific musical behaviours that might also inform the design of a high-level user interface; and finally

- identifying other uses for the ADTK, that might influence future development or provide avenues for further research.

To these ends, we conducted a study in which computer music practitioners were invited to submit sets of musical material suitable for arrangement-level performance, along with example performances created by themselves. For each participant, a selection of agents was designed (including manually designed ones, and ones using preset learning configurations) and used to generate new performances with
that participant’s musical material. The participant was then asked to listen to the agents’ performances and provide feedback about them.

In the following, we describe in detail how the study was carried out (Section 7.1). We then present the results in two parts. In the first part (Section 7.2), we describe the musical submissions (materials and examples) made by the participants; the participants’ responses to feedback questionnaires regarding the agents’ performances; and participants’ suggestions as to possible uses for the ADTK. In the second part (Section 7.3), we describe a selection of agent design techniques which were developed over the course of the study as agents were created for each participant. Finally, we discuss the results from various perspectives in Section 7.4.

7.1 Methods

We conducted a study in which participants were asked to rate the performances of a variety of musical agents. For each participant, the steps carried out were as follows:

1. The participant selected from their previous work, a composition in Ableton Live or Max with which they could perform live.

2. The participant created a set of example performances on which to train musical agents.

3. The participant described to the experimenter, the variables for controlling the Ableton Live set or Max patch, as well as the important features of the recorded examples.

4. The experimenter created a set of agents and used them to generate new performances with the Ableton Live set or Max patch.

5. The participant listened to the recorded performances and answered questions about them.

Further details are given in the following.
7.1.1 Participants

Seven participants were recruited through the mailing list of the Australasian Computer Music Association. The request for participants stipulated that they identified as ‘active computer music practitioners.’ No remuneration was offered and for this reason we stipulated that participants have ‘interest in (but not necessarily any experience of) applying algorithms in the context of computer music composition or performance.’ This was in order to increase the likelihood that participants would be sufficiently interested in the project to expend the significant time and effort to identify a composition, record example performances, and then give feedback on the resulting agent performances. Because the ADTK was implemented as a set of plug-ins for Max, we also stipulated that participants be Max or Ableton Live users. A short video illustrating the use of an early version of the ADTK in Ableton Live was included in the recruitment email.\footnote{The introductory video is available at: youtu.be/Jv78PTXdvWM.}

7.1.2 Recording and Post-Processing Example Performances

Here we describe the recording and post-processing of example performances in preparation for creating musical agents. Of the seven participants, one used a composition created in Max and those remaining used compositions created in Ableton Live. We describe the recording and post-processing related to Ableton Live compositions, followed by an account of these processes as they were carried out for the participant using Max.

Ableton Live

The Ableton Live users were supplied with the ADTK components necessary to record example performances (the Agent Designer and the Agent Designer Device). These were configured to store snapshots at the beginning of each bar, of the clips playing in each track, along with the values of any other parameters used to control the Ableton Live set. Thus for this study all agents created to control Ableton Live
sets operated with decision points occurring at the beginning of each bar. To assist participants in recording example performances, two short instructional videos were supplied in which the recording process was demonstrated\(^2\).

At the time of writing the ADTK has been designed exclusively for recording and controlling integer-valued variables in arrangement-level performance. Thus, additional steps were required to accommodate participants whose performances included the use of

- real-valued parameters, or
- the manipulation of parameters (real-valued or not) more than once per bar.

There were two such participants (see below). We refer to such real-valued control parameters, or parameters that are changed at arbitrary times during a performance—possibly continuously—as ‘gestural parameters’ (see Figure 7.1 for an example).

To accommodate the recording and control of gestural parameters, the following additional steps were taken before any agents were created. First, participants were asked to additionally record their examples using the built-in recording features of Ableton Live, which record timed sequences of clip changes and parameter value changes (recordings can be made using the Agent Designer and Ableton Live concurrently). Then, as a post-processing step, the experimenter manually

- enumerated the characteristic bar-length gestures found in the example performances (examples of such gestures are shown in Figure 7.2);
- created a mechanism whereby approximations of these gestures could be automatically sequenced, one bar at a time, according to integer-valued *gesture indices*\(^3\) (i.e. a number corresponding to each gesture); and finally
- added appropriate sequences of gesture indices to the data associated with each example performance.

\(^2\)The instructional videos are available at: youtu.be/Hc-nBOt5xIc and youtu.be/YJNCghVQ7bU.

\(^3\)The mechanisms for generating individual bar-length gestures were implemented as simple Max for Live devices.
These additional steps allowed agents to learn and produce sequences of gestural control data, even with decision points occurring only at the beginning of each bar. While this procedure was performed by hand, we envisage automating it in future versions of the ADTK (see Chapter 10 for further discussion of this).

One additional post-processing step was carried out for all recorded examples (regardless of the use of gestural parameters). Most examples included short portions at the beginning and end during which no sound was produced (e.g. after ‘record’ had been pressed, but before the participant had begun performing). These were removed.

Max

Max is an extremely open-ended platform, allowing for the development of a wide variety of performance environments, so it was not possible a priori to establish
7.1. Methods

Figure 7.2: Examples of parameter control gestures approximating those found in example performances. All gestures are 1 bar in length and the parameter is scaled to a range from 0.0 to 1.0 in each case. (a) A linear change from 0.1 to 0.4. (b) a linear change from 0.9 to 0.3. (c) A constant, but non-integer value (at 0.7). (d) an integer-valued parameter modulated from 0 to 1 for a short period during a bar.

a single procedure by which the ADTK could be incorporated into a Max patch. Instead, we planned to work directly with each Max user to integrate the ADTK into his/her patch. Since only one Max user participated, we give an outline here of the specific steps taken to adapt the Max patch in question, both for recording examples and for later control by ADTK agents.

The participant agreed that the musical output of the patch was without a discernible tempo and so there was no ‘natural’ interval by which to separate an agent’s decision points (or, correspondingly, by which to separate snapshots of parameter values). In addition, the participant’s performance style included quite rapid manipulation of certain parameters. To avoid the need to manually analyse gestures and to develop additional mechanisms to synthesize them (as was done for the gestural parameters described above), we decided to use a relatively short interval between decision points.

To arrive at an appropriate interval size, we recorded the participants’ performances by taking snapshots at a low sampling interval of 40 ms. We then reproduced the performances (by passing the control data back into the Max patch) using resampled versions of the recorded parameter data with progressively greater sampling intervals. With sampling intervals greater than approximately 400 ms it became
possible to hear differences between the original recorded performances (with control data sampled at 40 ms intervals) and the reproduced ones. Thus, we used an interval of 400 ms between decision points. Note that this is considerably faster than the interval between decision points for any of the Ableton Live agents (if we consider the interval between decision points to be equivalent to a four-beat bar, it corresponds to a tempo of 600 beats per minute).

For music without a clear tempo, we had previously found that regularly spaced decision points can result in a tempo being imposed on the music resulting from an agent’s performance; since frequently *some* change occurs at almost every decision point, the regularly-timed sequence of changes (disparate as they may be) is heard as a constant pulse or tempo (this was reported in [124]). To mitigate this, a mechanism was introduced to the Max patch whereby a randomly chosen delay was imposed on each parameter change, with the delay time between 0 and 380 ms (i.e. just shorter than the interval between decision points). As with the special treatment of gestural parameters in certain Ableton Live sets, we envisage introducing a feature similar to this in future versions of the ADTK (again, see Chapter 10 for more).

### 7.1.3 Agents

Six agents were created for each participant. Four were created according to predetermined learning configurations and two were designed by hand by the experimenter. Details are given in the following.

**The RANDOM Learning Configuration**

In the RANDOM learning configuration, each variable under the agent’s control was modelled with a uniform probability distribution. That is, at each decision point, a random value was drawn for each variable from a uniform probability distribution over the domain of the variable. Thus, the only way in which an agent of this type ‘learned’ from the example performances was to identify the domain of each variable. No rules were learnt and no custom variables were used. While RANDOM
agent’s of this type could be readily implemented in Max, for example, this learning configuration was included to provide a baseline with which the other five agent types might be compared.

**The PARALLEL Learning Configuration**

In the PARALLEL learning configuration, each variable under the agent’s control was modelled with an 8th-order VMM, learnt from the example performances. No rules were learnt and no custom variables were used, thus the VMMs operated in ‘parallel’. Again, it would be relatively straightforward to implement agents of this type in Max, and this learning configuration was included in order to empirically characterise the limitations of treating variables independently.

**The PRESET1 Learning Configuration**

The PRESET1 learning configuration used only VMMs and association rules to form a model of arrangement-level musical decision making. It was included primarily to investigate the models that would result from naïve use of these components without using custom variables for feature selection. In addition, we envisaged that, should this learning configuration prove useful, it could be included in a list of preset learning configurations available for selection in a high-level user interface for the Agent Designer.

Prior to creating each PRESET1 agent, the participant in question was asked to indicate which tracks in the Ableton Live set were ‘most structurally important’ to the musical performance. Our intention was to determine which tracks best defined the structure of the musical performances. For example, in techno music, the presence of a prominent kick drum beat can differentiate a high intensity section from a low intensity section, thus the kick drum track would be considered structurally important. In contrast, a track containing background sound effects used throughout the performance would not be considered structurally important. If the participant asked for clarification, it was suggested that those tracks most important to the
rhythmic and harmonic foundation to the music (if relevant) might also be most important to the musical structure. No limits were placed on the number of tracks that the participant could select. We envisaged that (again, should this strategy prove to be a promising one) this question could be presented to users of the Agent Designer as a preliminary step to using this learning configuration.

The \textsc{Preset1} learning configuration is shown in Table 7.1. It was created for training an agent to control an $N$ tracks and $M$ effects parameters in an Ableton Live set, after the user has selected $K$ of the $N$ tracks to be ‘most structurally important’ (as indicated above, the number $K$, as well as the selection of tracks, is determined by the user). Using this learning configuration, all variables were modelled using VMMs, with the maximum order set to 10 for the most structurally important tracks, and 1 for all other variables. All variables were included in a single rule group for association rule learning.

**The \textsc{Preset2} Learning Configuration**

The \textsc{Preset2} learning configuration used custom variables to model (i) the layering of different instruments (ii) metrical structure. Prior to commencing this study, informal testing had indicated that learning configurations of this type could be quite effective (details are given below). It was included as a second potential preset learning configuration that might be available for selection in a high-level user interface for the Agent Designer.

The \textsc{Preset2} learning configuration is shown in Table 7.2. Using this learning configuration, the highest priority variable was a custom variable, $c_0$, representing the number of Ableton Live tracks playing (the formula for $c_0$ has been explained previously: see the text surrounding Equation 5.16). The purpose of this custom variable was to model the way in which the number of instruments playing changes over time and thus to impose structure on the overall dynamics of the performance. It was modelled using a 5th-order VMM.

The next-highest priority variables were \textsc{Block} custom variables created for
Tables 7.1: The \textsc{Preset1} learning configuration for an agent intended to control \(N\) tracks and \(M\) audio effects parameters in an Ableton Live set. The variables are listed in priority order (see Section 5.4.5) with the highest-priority variable at the top. For each variable, the label, symbol and VMM order is shown. For Ableton Live track variables (i.e. variables indicating which clip is playing in a track), the letter \(t\) is used with the subscript indicating the track number. The tracks numbered \(1, \ldots, K\) are those indicated by the participant to be structurally important, whereas those numbered \(K + 1, \ldots, N\) are the remainder. For other variables that control audio effects, the letter \(p\) is used. For the rule group containing all variables, the the minimum support, \(S_{\text{min}}\), the minimum confidence, \(C_{\text{min}}\), and the maximum itemset size, \(\#_{\text{max}}\) (see Section 5.4.4) are shown; along with a list of the variables included in the group.
each track. These were intended to help model the short term metrical structure in each track. In Table 7.2, the block sizes are denoted $L_1, \ldots, L_N$ and these are the lengths of the longest clips found in tracks $1, \ldots, N$, respectively. These BLOCK custom variables reflect our supposition that the lengths of clips found in a particular track give an indication of the hypermetrical structure of the musical material (e.g. if four-bar drum loops are found on a given track then the track should be treated in blocks of four bars). The BLOCK variables were all modelled using 1st-order VMMs\footnote{Note that VMMs applied to BLOCK variables are used to draw new values once per block, rather than at each decision point. This loosely corresponds to a much higher-order model being applied to the track variable underlying the BLOCK custom variable.}. Finally, all remaining variables (individual track variables and effects parameters) were modelled using first-order VMMs. Note that in most cases the VMMs controlling individual track variables have no effect, since usually the values of these variables are completely determined by the higher-priority BLOCK variables.

For association rule learning, two rule groups were used. The first comprised the BLOCK variables and the second included all track variables and effect parameter variables. The BLOCK variables were treated separately since they completely determine the values of the track variables; including them with the track variables would have lead to the discovery of redundant rules. The purpose of the rule groups was to learn rules to capture the vertical stylistic constraints exhibited in the example performances.

**The DESIGN1 and DESIGN2 Learning Configurations**

The DESIGN1 and DESIGN2 learning configurations were designed by hand separately for each participant. They represented two efforts by the experimenter to create models that captured the style underlying the provided examples while allowing for significant variety in the agent’s performances. In each case, the designs were informed by descriptions provided to the experimenter of the musical material and the important aspects of the example performances. These descriptions were provided before any agents were created; there was no instance in which additional
### Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tracks Playing</td>
<td>$c_0 = \left[\sum_{\text{ANY}} (1, t_1), \sum_{\text{ANY}} (1, t_2), \ldots, \sum_{\text{ANY}} (1, t_N)\right]$</td>
<td>5</td>
</tr>
<tr>
<td>Track 1 Block</td>
<td>$c_1 = \left[\text{BLOCK}(L_1, t_1)\right]$</td>
<td>1</td>
</tr>
<tr>
<td>Track 2 Block</td>
<td>$c_2 = \left[\text{BLOCK}(L_2, t_2)\right]$</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Track N Block</td>
<td>$c_N = \left[\text{BLOCK}(L_N, t_N)\right]$</td>
<td>1</td>
</tr>
<tr>
<td>Track 1</td>
<td>$t_1$</td>
<td>1</td>
</tr>
<tr>
<td>Track 2</td>
<td>$t_2$</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Track N</td>
<td>$t_N$</td>
<td>1</td>
</tr>
<tr>
<td>Effects Param 1</td>
<td>$p_1$</td>
<td>1</td>
</tr>
<tr>
<td>Effects Param 2</td>
<td>$p_2$</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Effects Param M</td>
<td>$p_M$</td>
<td>1</td>
</tr>
</tbody>
</table>

### Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Variables</td>
<td>${S_{\text{min}}, C_{\text{min}}, #_{\text{max}}}$</td>
</tr>
<tr>
<td>Non-Block Variables</td>
<td>${0.05, 0.95, 4}$          ${c_1, \ldots, c_N}$</td>
</tr>
</tbody>
</table>
information was gathered from a participant after the agent design process had begun.

The DESIGN1 and DESIGN2 were included in the study in order to characterise the modelling capability of ADTK agents. They represent what can be achieved by an ‘expert’ using the software, since the author, as experimenter, was most familiar with the software and the modelling possibilities made possible by its features. The only restriction placed on the DESIGN1 and DESIGN2 designs was that it be possible to load the agent into the Fast Performer (i.e. create the BDD; see Chapter 6) in less than one minute. Within this limit, all designs were permitted. In keeping with the paradigm of interactive machine learning, changes to the training data set were also allowed (in some cases this made it easier to capture certain hypermetrical structures, for instance), though only minor changes were ever made (i.e. in cases where modifications were made, much fewer than 1% of the values in any one example were changed).

### 7.1.4 Agent Performances

Each agent was used to record three example performances. Since the Agent Designer contains no features for explicitly modelling musical form, each performance was truncated so that it ended at what was deemed by the experimenter to be musically appropriate point, insofar as was possible, with reference to the examples provided. For instance, rather than truncate a performance during a sustained portion of high intensity, it was allowed to continue until the intensity reduced (unless abrupt endings did occur in the example performances). In all but two cases, agent performances were no shorter than 80% of the duration of the shortest example provided by the participant and no longer than 120% of the duration of the longest example provided by the participant.

We emphasise that agent performances were not cherry-picked. In each case, the first three performances produced by an agent were those supplied to the participant. In addition, for the DESIGN1 and DESIGN2 agents, the design phase was
definitely completed before performances were generated; once the generation of performances had begun, no modifications were made to the design. Note, however, that performances were generated after each agent had been completed, thus, it was possible to inform the creation of the DESIGN2 agent by observing flaws that were revealed in the generated performances of the DESIGN1 agent. Thus, in some cases, the DESIGN2 agent was in part an iteratively improved version of the DESIGN1 agent.

7.1.5 Feedback Questionnaire

When a set of agent performances had been created, the participant was directed to a web-based questionnaire comprising eight separate pages. The pages were arranged as follows (details of the questions are given below). The first page contained brief instructions on completing the questionnaire; embedded audio players for playing the participant’s examples; and a set of three preliminary questions regarding the examples they had provided (see Figure 7.3). The following six pages were formatted identically, each corresponding to one of the six agents. Each contained a brief set of instructions; a set of embedded audio players for playing the agent performances; and a set of 14 questions about the performances (see Figure 7.4). The agent conditions were presented in a randomised order and the participant was not informed of the different conditions. The final page, comprised 6 additional questions in which the participant was asked to speculate about possible uses for the ADTK software, and invited to make general comments (see Figure 7.5). The participant was free to complete the questionnaire pages in any order, provided the final page was answered last. In addition the participant was free to revisit pages to replay performances. In particular, he/she could return to the first page to consult the example performances. Further details are presented in the following.

A list of the questions and definitions that comprised the questionnaire are given in Table 7.3. Since the 14 ‘Agent Questions’ (A1-A14) were presented six times (one for each agent condition), there were 93 questions in total with the final one, which
Figure 7.3: The first page of the feedback questionnaire.
Figure 7.4: An example of a questionnaire page for giving feedback on the performances of a particular agent. There were six of these pages in the questionnaire (one for each agent) presented in a random order.
invites general comments, being optional. Of the 92 compulsory questions, 20 were open-ended, allowing free-form text as answers. The remaining 72 were Likert-type questions, in which a statement was presented and the participant was required to indicate his/her level of agreement.

For the Likert-type questions, a 6-point scale was used with the options labelled as follows: 1=‘Disagree Strongly’, 2=‘Disagree Moderately’, 3=‘Disagree Slightly’, 4=‘Agree Slightly’, 5=‘Agree Moderately’, 6=‘Agree Strongly’. No neutral option (e.g. ‘neither agree nor disagree’) was included because we speculated that the task of judging agent performances was quite challenging and a participant might be drawn to such an option when uncertain. For consistency, we retained this set of options even for the less challenging Likert-type questions not related to agent performances.
7.1. Methods

Definitions

<table>
<thead>
<tr>
<th>Style Similarity:</th>
<th>Stylistically similar to the examples I provided.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musicality:</td>
<td>In a musical way, regardless of style similarity.</td>
</tr>
<tr>
<td>Vertical Structure:</td>
<td>Instantaneous combinations of musical material possibly including clips, effects parameters, ignoring the way combinations are sequenced.</td>
</tr>
</tbody>
</table>

Preliminary Questions

<table>
<thead>
<tr>
<th>Preliminary Questions</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>There is strict short-term metrical structure in my examples (e.g. loops that are four bars long and should not be broken mid-loop, loops that must be kept in synch with each other).</td>
</tr>
<tr>
<td>I2</td>
<td>There is clear long-term musical structure in my examples.</td>
</tr>
<tr>
<td>I3</td>
<td>Please write a few words indicating the style or genre of the musical material and examples you provided.</td>
</tr>
</tbody>
</table>

Agent Questions

<table>
<thead>
<tr>
<th>Agent Questions</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Overall, the agent performs with style similarity.</td>
</tr>
<tr>
<td>A2</td>
<td>Overall, the agent performs with musicality.</td>
</tr>
<tr>
<td>A3</td>
<td>The long-term structure of the performance shows style similarity.</td>
</tr>
<tr>
<td>A4</td>
<td>The long-term structure of the performance shows musicality.</td>
</tr>
<tr>
<td>A5</td>
<td>The short-term structure of the performance shows style similarity.</td>
</tr>
<tr>
<td>A6</td>
<td>The short-term structure of the performance shows musicality.</td>
</tr>
<tr>
<td>A7</td>
<td>The vertical structure of the performance shows style similarity.</td>
</tr>
<tr>
<td>A8</td>
<td>The vertical structure of the performance shows musicality.</td>
</tr>
<tr>
<td>A9</td>
<td>Regardless of the agent’s style similarity or musicality, its performances could give me new musical ideas for using this musical material.</td>
</tr>
<tr>
<td>A10</td>
<td>I can hear the agent mingling different aspects of my examples in its performance.</td>
</tr>
<tr>
<td>A11</td>
<td>Regardless of style similarity or musicality, the agent seems capable of more variety than was exhibited in my examples.</td>
</tr>
<tr>
<td>A12</td>
<td>The main thing I hear the agent doing wrong is:</td>
</tr>
<tr>
<td>A13</td>
<td>The main thing I hear the agent doing right is:</td>
</tr>
<tr>
<td>A14</td>
<td>Please include any other comments that you think might be relevant.</td>
</tr>
</tbody>
</table>

Final Questions

<table>
<thead>
<tr>
<th>Final Questions</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>This software could be used to create systems which generate ambient music.</td>
</tr>
<tr>
<td>F2</td>
<td>This software could be used to create systems which generate sound for art installations.</td>
</tr>
<tr>
<td>F3</td>
<td>This software could be used to create systems which generate classical music.</td>
</tr>
<tr>
<td>F4</td>
<td>This software could be used to create systems which generate pop music.</td>
</tr>
<tr>
<td>F5</td>
<td>Can you suggest one possible use of the software, not mentioned above, that you think might be successful?</td>
</tr>
<tr>
<td>F6</td>
<td>(Optional) If you would like to elaborate on any of your answers above, or make any other comments, please do so here.</td>
</tr>
</tbody>
</table>

Table 7.3: A complete list of the definitions and questions presented to participants to gather feedback on agent performances. The ‘Agent Questions’ were presented separately for each of the six agents. The ‘Answer’ column indicates whether the participant was required to give a Likert-style response (Likert) or an open-ended answer (Text).
7.2 Results: Participant Submissions and Responses

7.2.1 Compositions and Examples Submitted

In this section, we describe the compositions and examples submitted for this study. The purpose is to characterise the requirements of an agent’s decision making behaviour in each case. The descriptions presented have been informed by discussions with each participant before the agent designs were undertaken, examination of the example performances supplied, and finally, each participant’s responses to the preliminary questions in the feedback questionnaire (see Table 7.3, above). However, note that the last of these was not available before the agents were designed. Finally, to support the text, one example performance from each participant has been made available to listen to online.

Materials Submitted by Participant P1

Participant P1 selected a composition in Ableton Live which he/she described as ‘Electro-acoustic, vocal, improvised, bass-driven’ (question I3; see Table 7.3) and in discussion of the material, also described it as ‘ambient’. He/she submitted three example performances, with durations 5’45”, 6’24” and 7’02” and comprising a total of 480 decision points (i.e. musical bars). A list of the variables used to control the Ableton Live set is shown in Table 7.4.

The participant strongly disagreed that there was clear long term structure in the example performances (question I2), yet there were distinct musical sections. In particular, the example performances were characterised by alternating sections underpinned by a stable rhythmic material (a ‘solid groove’) and breakdown sections distinguished primarily by their lack of a stable rhythm and more frequent use of the Effected Percussion tracks. During the rhythmic sections, the Main Drums track often began with a brief exposition of some less rhythmically intense material, followed by a prolonged period using the most rhythmically intense material, followed in turn by

---

5See: am-process.org/thesis-examples.
### 7.2. Results: Participant Submissions and Responses

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Drums</td>
<td>8 clips</td>
<td>Clip lengths: 2, 4 bars</td>
</tr>
<tr>
<td>Effected Percussion 1</td>
<td>5 clips</td>
<td>Clip lengths: 1, 4, and 8 bars</td>
</tr>
<tr>
<td>Effected Percussion 2</td>
<td>5 clips</td>
<td>Clip lengths: 2, 4, and 8 bars</td>
</tr>
<tr>
<td>Bass</td>
<td>4 clips</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Synthesizer Loops 1</td>
<td>7 clips</td>
<td>Clip lengths: 2, 4, 8, and 16 bars</td>
</tr>
<tr>
<td>Synthesizer Loops 2</td>
<td>9 clips</td>
<td>Clip lengths: 4, and 8 bars</td>
</tr>
<tr>
<td>Vocal Samples 1</td>
<td>8 clips</td>
<td>Clip lengths: 1, 2, 3, 5, 7 and 8 bars</td>
</tr>
<tr>
<td>Vocal Samples 2</td>
<td>4 clips</td>
<td>Clip lengths: 2, 4, 5 and 8 bars</td>
</tr>
<tr>
<td>Echo Effect</td>
<td>on/off</td>
<td>Echo effect applied to Vocal Samples 1 Track</td>
</tr>
</tbody>
</table>

Table 7.4: Summary of the Ableton Live set used by P1. There are eight Ableton Live tracks and one effects parameter. For each of the tracks, the ‘value’ column gives the number of clips and the ‘description’ column gives the variety of clip durations. For the effects parameter, the ‘value’ column gives the possible settings (here: on or off).

A brief reduction in intensity before stopping completely to transition to a breakdown.

Long term structure was also imposed by the bass instrument alternating between periods during which the first three loops were used (these were variations of the same material) and periods during which the fourth loop was used (this was distinctly different). However, changes between the two different bass patterns (clips 1-3, and clip 4) did not coincide with changes in section related to the main drums.

The participant slightly disagreed that there was strict metrical structure in the example performances (question I1), yet the bass loops, main drum loops and certain synthesizer loops adhered to a four-bar hypermetrical structure (i.e. they were aligned in four-bar sections). However, much material did not require metrical alignment; the synthesizer and vocal clips were layered and juxtaposed in a variety of ways throughout the performance. Within these combinations, there are certain regular patterns such as particular sequences of vocal samples, and the certain correspondences between particular synthesizer and bass loops. In addition, certain combinations of percussive material were avoided. Finally, during each of the example performances, the echo effect was turned on once and left on for a substantial period (e.g. 20 bars or more), as a source of subtle variety.
Materials Submitted by Participant P2

Participant P2 used a composition implemented as a Max patch comprising six separate instruments for producing abstract, textural output by processing pre-loaded audio samples. All of the audio samples were short extracts from the song, ‘Last Train to Clarksville’ by The Monkees, however, in the example performances they were largely not recognisable as such. He/she described the musical material in the following way (question I3):

‘The musical examples provided would be classified as free-form improvisation with the caveat that the musical material within the software is fixed. While the software for processing the material remains the same for all the examples the scope for experimentation is vast. Decisions regarding which material and transformative processes to focus on are momentary, the structure and soundworld emerging in an organic fashion, relying on intuition, experience and actively listening to and engaging with the resultant sonic material.’

Consistent with this, the participant strongly disagreed that there was strict short term metrical structure, and moderately agreed that there was clear long-term structure in the example performances (questions I1 and I2, respectively). He/she submitted four example performances, with durations 4’02”, 3’31”, 3’52” and 4’40” comprising a total of 2412 decision points. This large number of decision points, in comparison to the participants that used Ableton Live, is due to the relatively fast sampling of variable control values (once every 400 ms). The variables for controlling the Max patch are shown in Table 7.5.

All of the example performances began with a single instrument playing. In three of the four examples, Instrument 1 was introduced first and remained on for the entire performance (though its output did not remain constant). In the same three examples, Instrument 2 was introduced shortly after the beginning and remained on for most of the remainder of the performance. The other four instruments were brought in
<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument 1 Level</td>
<td>0-3</td>
<td>Volume control (0 = off)</td>
</tr>
<tr>
<td>Instrument 1 Transposition</td>
<td>0-3</td>
<td>Pitch control</td>
</tr>
<tr>
<td>Instrument 1 Position</td>
<td>0-3</td>
<td>Part of sample being processed</td>
</tr>
<tr>
<td>Instrument 1 Scanning Speed</td>
<td>0-3</td>
<td>Speed of scanning through sample</td>
</tr>
<tr>
<td>Instrument 1 Filter</td>
<td>0-2</td>
<td>Filter setting</td>
</tr>
<tr>
<td>Instrument 2 Level</td>
<td>0-3</td>
<td>Volume control (0 = off)</td>
</tr>
<tr>
<td>Instrument 2 Transposition</td>
<td>0-3</td>
<td>Pitch control</td>
</tr>
<tr>
<td>Instrument 2 Position</td>
<td>0-3</td>
<td>Part of sample being processed</td>
</tr>
<tr>
<td>Instrument 2 Scanning Speed</td>
<td>0-3</td>
<td>Speed of scanning through sample</td>
</tr>
<tr>
<td>Instrument 2 Filter</td>
<td>0-2</td>
<td>Filter setting</td>
</tr>
<tr>
<td>Instrument 3 Level</td>
<td>0-3</td>
<td>Volume control (0 = off)</td>
</tr>
<tr>
<td>Instrument 3 Transposition</td>
<td>0-3</td>
<td>Pitch control</td>
</tr>
<tr>
<td>Instrument 3 Position</td>
<td>0-3</td>
<td>Part of sample being processed</td>
</tr>
<tr>
<td>Instrument 3 Density</td>
<td>0-4</td>
<td>Density of sound grains</td>
</tr>
<tr>
<td>Instrument 4 Level</td>
<td>0-3</td>
<td>Volume control (0 = off)</td>
</tr>
<tr>
<td>Instrument 4 Transposition</td>
<td>0-3</td>
<td>Pitch control</td>
</tr>
<tr>
<td>Instrument 4 Position</td>
<td>0-3</td>
<td>Part of sample being processed</td>
</tr>
<tr>
<td>Instrument 4 Density</td>
<td>0-4</td>
<td>Density of sound grains</td>
</tr>
<tr>
<td>Instrument 5 Level</td>
<td>0-3</td>
<td>Volume control (0 = off)</td>
</tr>
<tr>
<td>Instrument 5 Transposition</td>
<td>0-3</td>
<td>Pitch control</td>
</tr>
<tr>
<td>Instrument 5 Scanning Speed</td>
<td>0-3</td>
<td>Speed of scanning through sample</td>
</tr>
<tr>
<td>Instrument 6 Level</td>
<td>0-3</td>
<td>Volume control (0 = off)</td>
</tr>
<tr>
<td>Instrument 6 Transposition</td>
<td>0-3</td>
<td>Pitch control</td>
</tr>
<tr>
<td>Instrument 6 Scanning Speed</td>
<td>0-3</td>
<td>Speed of scanning through sample</td>
</tr>
</tbody>
</table>

Table 7.5: Summary of the Max Patch used by P2. There were six instruments, each of which processed a short audio sample. There were three types of instruments in total, with two instances of each. That is, Instrument 1 and Instrument 2 were identical, but they were loaded with different audio samples, and similarly for 3 and 4, etc.
and out frequently (sometimes almost synchronously) in various combinations. The fourth example was similar to the other three in that instruments were frequently brought in and out in various combinations. However, instruments 1 and 2 did not form a foundation; there were periods of silence during the performance. Finally, certain patterns were used repeatedly in the performances. These included quickly raising and lowering the level of an instrument over a period of approximately between two and five seconds, and quickly raising and lowering the pitch of an instrument over a similar period, in order to introduce dynamic and melodic motifs, respectively.

**Materials Submitted by Participant P3**

Participant P3 selected a techno composition in Ableton Live from a catalogue of previous work. It featured a number of musical devices common to techno music, such as the use of sparse instrumentation, a high-pass filter applied to the kick drum used intermittently to cut the low frequencies in order to build tension, and the gradual increasing of the cutoff frequency on a low-pass filter applied to melodic material, as part of a buildup in intensity. He/she described the musical material and the example performances as ‘loop-based techno’ (question I3) and strongly agreed that there was short-term metrical structure and clear long-term structure in the example performances (questions I1 and I2, respectively). The set of examples comprised four performances with durations of 4’00”, 3’42”, 4’25” and 4’25”, totalling 540 decision points (i.e. bars). A summary of the control variables used for the Ableton Live set is shown in Table 7.6.

All performances began with just the Main Beats and Bass tracks playing and in all cases the Main Beats track was active throughout (though different clips were used), and the Bass was active for most of the performance (sometimes stopping a little before the end). In some cases, the beginning was also characterised by the use of one or more effects on the Main Beats track. In all cases, the remaining three instruments were gradually introduced either one at a time, or in various
Table 7.6: Summary of the Ableton Live set used by P3. There are five Ableton live tracks. In each case the variety of different clip lengths is given in the third column. There were five effects parameters, two of which were gestural parameters (see Section 7.1.2).

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Beats</td>
<td>4 clips</td>
<td>Clip lengths: 1,4</td>
</tr>
<tr>
<td>Bass</td>
<td>1 clip</td>
<td>Clip lengths: 1</td>
</tr>
<tr>
<td>Synth Chords 1</td>
<td>4 clips</td>
<td>Clip lengths: 2,4</td>
</tr>
<tr>
<td>Synth Chords 2</td>
<td>4 clips</td>
<td>Clip lengths: 2,4</td>
</tr>
<tr>
<td>Other Percussion</td>
<td>2 clips</td>
<td>Clip lengths: 1</td>
</tr>
<tr>
<td>Reverb</td>
<td>on/off</td>
<td>Reverb effect on Main Beats track</td>
</tr>
<tr>
<td>Low Cut</td>
<td>on/off</td>
<td>High pass filter or the Main Beats track</td>
</tr>
<tr>
<td>Beat Repeat</td>
<td>on/off</td>
<td>Rhythmic ‘stutter’ effect on Main Beats track</td>
</tr>
<tr>
<td>Cutoff</td>
<td>Gestural</td>
<td>Cutoff frequency of filter applied to the two Synth Chords tracks</td>
</tr>
<tr>
<td>Resonance</td>
<td>Gestural</td>
<td>Resonance of filter applied to the two Synth Chords tracks</td>
</tr>
</tbody>
</table>

combinations. Instruments were always introduced after multiples of eight bars. Usually, as the Synth Chords instruments were introduced, the cutoff frequency of the lowpass filter applied to those instruments was gradually increased over a period of eight or sixteen bars.

Once the initial buildup was complete, performances broadly took the following structure (though there were variations). First, there was a sustained period of high intensity followed by a brief breakdown, followed in turn by a second, shorter, high-intensity period and then a short section similar to a breakdown to finish. The breakdowns were characterised primarily by the use of one of the less rhythmically intense loops on the Main Beats tracks and/or the use of the effects on that track. When these effects were used, the Reverb and Low Cut effects on the Main Beats track were always used together in all cases (i.e. they were always turned on at the same time). Finally, there was clear hypermetrical structure: the loops in the Synth Chords tracks were metrically synchronized throughout; indeed almost all clip changes occurred after multiples of four bars, and effects were often used for four-bar blocks.
Materials Submitted by Participant P4

Participant P4 selected a techno composition in Ableton Live. The participant strongly agreed that there was short-term metrical structure in the example performances and slightly agreed that there was clear long-term structure in the example performances (questions I1 and I2, respectively). He/she described the composition and examples as follows (question I3):

‘techno. starts with a sample of an audience member shouting “Go on Jeff Mills” at a night club in Dublin in 2012. Drums sounds come from a Roland TR-909 emulator. The track features a 4/4 bass drum, snare fills, and a loud slightly distorted lead melody which comes in and out. the loops are sometimes shortened and lengthened during the performance, broken down and built back up, but the 4/4 time signature usually remains.’

The participant submitted four example performances, with durations of 1’58”, 1’47”, 1’52”, 2’14” and a total of 249 decision points (i.e. bars). A list of the variables controlling the Live set is given in Table 7.7.

Each example performance begins with the vocal sample referred to above, which is played on the Single Vocal Sample track. This is then followed directly by looping clips, which are short segments of the same sample, in the Vocal Sample Loops track. These looping clips continue in each case for much of the remainder
of the performance and thus, a single vocal sample forms the basis for the entire composition.

Broadly, the example performance were characterised by the following elements. A buildup section after the initial exposition of the vocal sample, during which instruments are introduced either individually or in combinations, with changes usually separated by multiples of two bars. A high-intensity portion, characterised by polyrhythms arising from the looping of clips of different lengths, and punctuated by frequent one- or two-bar interruptions which are dominated by rapidly looping clips in the Vocal Sample Loops track. A breakdown to one or two tracks and the reintroduction of instruments (in some cases with previously unused clips). An ending section during which instruments are removed over a short period, sometimes with a replay of the full vocal sample as a coda.

**Materials Submitted by Participant P5**

Participant P5 selected a composition in Ableton Live that he/she stated would ‘probably fall into the loose genre of ambient/techno’ (question I3). He/she strongly agreed that there was strict short-term metrical structure and moderately agreed that there was clear long-term structure in the examples (questions I1 and I2, respectively). The participant submitted four examples, with durations of 4’27”, 4’11”, 4’35”, 4’27” comprising a total of 512 decision points.

The variables for controlling the Ableton Live set are listed in Table 7.8, which shows clips lengths consistent with the four- and eight-bar hypermetrical structure of the performances. In addition, there were a number of unusually long clips. However, each of the long clips contained a number of repetitions of a four- or eight-bar phrase with subtle timbral variations applied to each of the repetitions. Thus, each of these long clips could be stopped before they came to an end, but only in accordance with the four- or eight-bar structure of the material contained within.

Each example performance began with the Rhythmic Background track playing in isolation and this track continued in each case for the duration of the performance.
### Table 7.8: Summary of the Ableton Live set used by P5. All variables are Ableton live tracks. In each case the variety of different clip lengths is given in the third column.

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kick Drum</td>
<td>2 clips</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Closed Hi-Hat</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Open Hi-Hat</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Cymbal</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Open Hi-Hat 2</td>
<td>1 clip</td>
<td>Clip lengths: 8 bars</td>
</tr>
<tr>
<td>Tom</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Clap and Snare</td>
<td>2 clips</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Bass</td>
<td>5 clips</td>
<td>Clip lengths: 4, 16, 32 bars</td>
</tr>
<tr>
<td>Ambient Guitar 1</td>
<td>1 clip</td>
<td>Clip lengths: 8 bars</td>
</tr>
<tr>
<td>Ambient Guitar 2</td>
<td>1 clip</td>
<td>Clip lengths: 32 bars</td>
</tr>
<tr>
<td>Synth Melody</td>
<td>3 clips</td>
<td>Clip lengths: 48 bars</td>
</tr>
<tr>
<td>Rhythmic Background</td>
<td>1 clip</td>
<td>Clip lengths: 132 bars</td>
</tr>
</tbody>
</table>

In addition, the example performances were characterised by the gradual introduction of material in various combinations with new instruments being added after multiples of eight bars. Generally, pitched instruments (the Bass, Ambient Guitar and Synth Melody tracks), once introduced, would remain playing until they were removed as the performance came to an end. The percussion tracks, on the other hand, were layered in various ways to create dynamic variation and tension throughout; with changes usually occurring after multiples of four or eight bars. The sequencing of the Bass track was similar in all performances, with the five clips being used in the same order each time (though with varying durations).

**Materials Submitted by Participant P6**

Participant P6 selected an Ableton Live composition that he/she described as ‘mid-tempo electronica with dub and trip hop influences’ (question I3). He/she strongly agreed that there was short-term metrical structure and slightly agreed that there was clear long-term structure, in the example performances (questions I2 and I2, respectively). The participant submitted four example performances, with durations 2’48”, 2’56”, 2’52”, 3’04” comprising a total of 326 decision points. Table 7.9 shows a list of the variables used to control the Ableton Live set.
## Results: Participant Submissions and Responses

Table 7.9: Summary of the Ableton Live set used by P6. There were 14 Ableton Live tracks. For each of these, the variety of different clip lengths is given in the third column. There were nine effects parameters, all of which were gestural parameters; despite being discrete-valued parameters (on/off switches for various effects units) they were frequently manipulated faster than once per bar (see Section 7.1.2).

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound Effects</td>
<td>3 clips</td>
<td>Clip lengths: 2, 4 bars</td>
</tr>
<tr>
<td>Bell Melody</td>
<td>2 clips</td>
<td>Clip lengths: 4, 6 bars</td>
</tr>
<tr>
<td>Main Beat</td>
<td>2 clips</td>
<td>Clip lengths: 2 bars</td>
</tr>
<tr>
<td>Ambient Percussion 1</td>
<td>1 clip</td>
<td>Clip lengths: 2 bars</td>
</tr>
<tr>
<td>Bass</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Synth Chords</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Synth Melody</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Claps 1</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Kick Drum 1</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Claps 2</td>
<td>1 clip</td>
<td>Clip lengths: 4 bars</td>
</tr>
<tr>
<td>Ambient Percussion 2</td>
<td>1 clip</td>
<td>Clip lengths: 6 bars and 1 beat</td>
</tr>
<tr>
<td>Synth Solo</td>
<td>2 clips</td>
<td>Clip lengths: 4, 21 bars</td>
</tr>
<tr>
<td>Electric Piano</td>
<td>1 clip</td>
<td>Clip lengths: 8 bars</td>
</tr>
<tr>
<td>Kick Drum 2</td>
<td>1 clip</td>
<td>Clip lengths: 3 bars</td>
</tr>
<tr>
<td>Master Effect 1</td>
<td>Gestural</td>
<td>Chorus-type effect on Master Track</td>
</tr>
<tr>
<td>Master Effect 1</td>
<td>Gestural</td>
<td>Filter effect on Master Track</td>
</tr>
<tr>
<td>Master Effect 3</td>
<td>Gestural</td>
<td>‘Stutter’ effect on Master Track</td>
</tr>
<tr>
<td>Melody Effect 1</td>
<td>Gestural</td>
<td>Echo effect on Bell Melody Track</td>
</tr>
<tr>
<td>Melody Effect 2</td>
<td>Gestural</td>
<td>Echo effect on Bell Melody Track</td>
</tr>
<tr>
<td>Main Beat Effect 1</td>
<td>Gestural</td>
<td>‘Stutter’ effect on Main Beat Track</td>
</tr>
<tr>
<td>Main Beat Effect 2</td>
<td>Gestural</td>
<td>‘Stutter’ effect on Main Beat Track</td>
</tr>
<tr>
<td>Electric Piano Effect 1</td>
<td>Gestural</td>
<td>Vocoder effect on Electric Piano Track</td>
</tr>
<tr>
<td>Electric Piano Effect 2</td>
<td>Gestural</td>
<td>‘Stutter’ effect on Electric Piano Track</td>
</tr>
</tbody>
</table>
Central to the example performances is a four-bar repeating chord progression carried primarily by the *Synth Chords* and *Electric Piano* tracks. One or both of these instruments is playing most of the time, and instruments are layered on top in various ways with changes usually happening after multiples of four bars. Sections characterised by large numbers of tracks abruptly give way to four- or eight-bar sections in which only a few tracks are playing. Such breakdowns often coincide with the introduction of new material (for instance swapping the *Synth Chords* for the *Electric Piano*) and they are often anticipated by ‘fills’ achieved, for example, by turning on one or more effects (particularly ‘stutter’ effects) for the final bar before the change.

The four-bar hypermetrical structure is emphasised in a number of ways. In some cases fills occur on the fourth bar (though not followed by an abrupt change), and other devices are also used, such as the use of single percussive samples in the *Sound Effects* track, at the beginning of a four-bar loop; and the activation of the vocoder effect on the electric piano track for one four-bar block. There are clips in the Ableton Live set with lengths that are not integer multiples or divisors of four bars, however, in all cases these contain background material and the odd length serves only to add subtle variations to the sound.

**Materials Submitted by Participant P7**

Participant P7 used a textural, electroacoustic composition in Ableton Live. He/she described it in the following way (question I3):

‘The style of the examples is ambient, electro-acoustic music. The examples evolve slowly, and although most of the material loops, there is no concern for creating discernible rhythmic structures.’

Consistent with this, he/she disagreed strongly that there was short-term metrical structure, and agreed slightly, that there was clear long-term structure in the example performances (question I1 and I2, respectively). The participant submitted four
## 7.2. Results: Participant Submissions and Responses

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granular Texture 1</td>
<td>2 clips</td>
<td>Clip lengths: 9.5 s</td>
</tr>
<tr>
<td>Granular Texture 2</td>
<td>2 clips</td>
<td>Clip lengths: 10 s</td>
</tr>
<tr>
<td>Granular Texture 3</td>
<td>2 clips</td>
<td>Clip lengths: 19 s</td>
</tr>
<tr>
<td>Granular Texture 4</td>
<td>2 clips</td>
<td>Clip lengths: 32.5 s</td>
</tr>
<tr>
<td>FM Timbres</td>
<td>2 clips</td>
<td>Clip lengths: 14 s</td>
</tr>
<tr>
<td>Foley Samples</td>
<td>3 clips</td>
<td>Clip lengths: 7.5 s, 8 s, 16 s</td>
</tr>
<tr>
<td>Abstract Texture</td>
<td>2 clips</td>
<td>Clip lengths: 16 s</td>
</tr>
<tr>
<td>Additive Synth Timbres 1</td>
<td>2 clips</td>
<td>Clip lengths: 41.5 s</td>
</tr>
<tr>
<td>Additive Synth Timbres 2</td>
<td>2 clips</td>
<td>Clip lengths: 64 s</td>
</tr>
</tbody>
</table>

Table 7.10: Summary of the Ableton Live set used by P7. All variables are Ableton live tracks. In each case the variety of different clip lengths is given in the third column.

Example performances, with durations 3’28”, 3’32”, 3’48” and 3’36” comprising a total of 425 decision points. Since the musical material was entirely without a discernible tempo or rhythm, the 120 BPM tempo of the Ableton Live set is only nominal, and serves to simply set the decision point times to regularly occur every two seconds.

Table 7.10 shows the variables used to control the Ableton Live set. Since the units of bars and beats are not relevant in this case, clip lengths are given in seconds. In eight of the tracks (all except Foley Samples), there were two clips and in each case the first of these contained the main sound material. Each clip of main sound material contained an amplitude envelope so that the sound would slowly fade in when the clip was triggered, and then a sustained portion would loop until the clip was stopped. To end the material the second clip was triggered. In each case, this contained more of the same sound material, but with an amplitude envelope applied so that it faded out gradually. Thus, material in these tracks was introduced by triggering the first clip, and removed by triggering the second one (which did not loop), rather than simply stopping the track abruptly. The exception to this was the Foley Samples track, which contained four separate samples (non-looped) that were used sporadically throughout.

The example performances were characterised by the slow layering and juxtaposition of different textural material, with sustained portions punctuated by sounds on
the *Foley Samples* track. Each performance began with only a single track playing (usually *Additive Timbres 2*) and ended with only one or two tracks playing. In addition, the performances rarely became very dense; usually no more than five tracks were playing at once.

### 7.2.2 Participant Responses

In this section, we present the participants’ responses to the feedback questionnaire in order to empirically characterise the six agent types. We will refer to the statements for the four Likert-type questions related to style similarity (A1, A3, A5 and A7; see Table 7.3) collectively as the ‘similarity statements’, and for the four questions related to musicality (A2, A4, A6 and A8) collectively as the ‘musicality statements’.

Audio examples of selected agents have been made available online\(^6\). For each participant, a performance generated with his/her most positively judged agent is available, along with a performance by one of the more poorly judged agents. In each case, the latter was chosen to highlight particular aspects relevant to the participant’s responses. Each agent for which a performance is available as audio will be marked with a dagger symbol (**†**) the first time it is mentioned.

**Participant P1 Responses**

The responses submitted by participant P1 are shown in Figure 7.6. For the RANDOM† agent, the participant indicated at most slight agreement with any of the similarity statements. In contrast, for the PRESET2† agent, the participant indicated strong agreement with all of the similarity statements. For the other four agents, the responses to these statements lie in between these two extremes.

The responses to the musicality statements follow a similar trend to those for the similarity statements. For the RANDOM agent, the participant slightly disagreed with three of the four musicality statements and slightly agreed with the fourth (vertical musicality). Overall, the highest levels of agreement to statements related

---

6See: am-process.org/thesis-examples.
to musicality, were indicated for the PRESET1 agent (three strong agreements and one moderate agreement) and the PRESET2 agent (three strong agreements and one slight agreement). The responses for the other four agents are in between these two extremes, though they are perhaps closer to those for the two PRESET agents than those for the RANDOM agent.

Participant P1 indicated moderate or strong agreement for all agents with regard to the potential for generating new ideas (A9) and the mingling of example performances (A10). He/she indicated strong agreement that the PRESET2 agent exhibited more variety than in the example performances (A11), and slight agreement with this statement for the RANDOM and PARALLEL agents (moderate agreement for the remaining agents).

The written comments given by the participant (questions A12-A14) for the six agents are broadly consistent with the responses reported above. In the case of the RANDOM agent, the low levels of agreement with the musicality statements are reflected in the written comments:

‘a bit disjointed with sudden and quick changes that don’t always seem to follow a logical order. Sometimes there is a bit of dissonance between the synthesiser parts and the vocal parts seem to get cut off mid-phrase.’
Its potential for generating new ideas is also acknowledged (‘some really great rhythmic patterns’, ‘new ideas are being suggested to me’). In addition, consistent with the generally positive judgements of the PRESET2 agent, the participant writes that the agent ‘seems to be performing a more imaginative mix of my samples ... a good combination of the expected and unexpected.’

There is one comment that appears inconsistent with the Likert-type responses. As noted above, the PRESET1 agent was judged overall to exhibit the highest levels of musicality, yet of this agent’s performances the participant wrote:

‘There are a couple of gaps that don’t seem to have any particular function. The agent doesn’t do builds and breakdowns as a human would. Sometimes the changes seem arbitrary.’

However, these criticisms give no indication of the importance attached to them by the participant, and he/she may not have judged them of sufficient importance to reduce his/her levels of agreement to any of the Likert-type questions. This explanation accounted for a similar apparent inconsistency in the case of another participant (P2, see below).

Finally, we note that while the participant indicated moderate or strong agreement for every agent with respect to its potential for generating new ideas, the types of ideas may not be the same in each case. For example the RANDOM agent was praised for short-term musical ideas such as ‘great rhythmic patterns’ and ‘dramatic pauses’. The PARALLEL agent was praised for interesting—but not necessarily short-term—ideas such as ‘isolated vocal parts’ and the ‘mingling of filtered and non-filtered drum parts ... that goes well with the vocal’. These contrast with comments on the PRESET1, DESIGN1 and DESIGN2 agents which relate to higher-level musical ideas (‘sounds more minimal and spacious than my own mixes’, ‘surprised and impressed by the “creativity” of the agent, particularly the ordering of loops’, ‘creating a dreamy atmosphere’, ‘I like the way it suggests new structures and dynamics’).
7.2. Results: Participant Submissions and Responses

Participant P2 Responses

Taken in isolation, the Likert-type responses provided by participant P2 indicate indiscriminately positive judgements of the agent performances: To nine of the eleven Likert-type questions, the participant responded with moderate or strong agreement for all six agents (see Figure 7.7). Moreover, some responses appear at odds with the written comments, which included detailed characterisations of the agents’ performances including flaws of various types. For example, for the DESIGN1 agent, the participant responded with strong agreement to all similarity statements, yet he/she commented ‘There are durational and gestural aspects that I feel the agents [sic] do not judge in a manner similar to what I would do.’

A follow-up question was sent to this participant requesting some elaboration with regard to his/her approach to answering the Likert-type questions. In response, he/she wrote:

‘I found the agents performances to be very good and any flaws I highlighted where [sic] quite trivial. It was difficult to find fault ... Part of this is due to the nature of the music itself ... Who is to say what is right or wrong in improvisation?’

In addition the participant considered the agent performances very much from the
point of view of a live improviser, sympathetic to the challenges of performing live electronic music:

‘Also it’s one thing to listen back to a performance and point out areas where you feel that things could have been tighter or changed in some way. In the middle of a performance the decision making process is working flat out taking into account lots of different aspects all at once. If something goes on too long this could be because you are busy preparing something new to happen or at the time it feels the appropriate length.’

Thus, though the participant did not consider the agent performances to be perfect, he/she did not see the flaws were as hugely significant, particularly in light of the envisaged improvisatory context. In addition, it seems likely that the performance style that the participant had in mind included few, if any, strict constraints with regard to how the material could be used (he/she did not articulate any in particular). Therefore, for this participant we rely primarily on the written comments to characterise the different agents.

Of the RANDOM agent, the participant wrote that ‘Structurally it had some flaws at times but in general it was very good’, and noted its effectiveness at creating ‘a mass of sonic material, constantly bubbling away’ and ‘a more varied and denser maelstrom of sonic material’ than the participant did in the example performances. The participant judged this agent most positively with regard to the variety exhibited in its performances.

The PARALLEL† agent was described as executing ‘gestures ... similar in style and substance’ to those in the example performances. Yet, there were structural flaws:

‘The agents [sic] do not use silence in the same manner [as in the example performances]. They introduce a more continuous background of sound ... I feel that silence plays an integral role in the original performance and this is something that I felt the agents didn’t capture.’

In addition, certain combinations of parameters that arose in the agent performances
caused the sample manipulation instruments (see Section 7.2.1) to reveal more of the original samples being processed, whereas the example performances ‘showed the material in its more abstract guise, never really introducing the original material, instead concentrating more so on the outer edges of pitch.’

Of the PRESET1 agent performances, participant P2 wrote that ‘The agents [sic] mimic some of my musical gestures and vertical layer [sic] with a high degree of similarity.’ However, again, there were structural flaws:

‘I find the agents [sic] in these examples musically erratic at times. Some gestures jump too quickly creating a jarring sense of continuity. This makes for a less satisfying overall structure. the [sic] examples in this instance do not flow from beginning to end.’

These flaws may contribute to the participant’s moderate agreement (as opposed to strong agreement) with the statements relating to long-term and short term style similarity and musicality.

In contrast to the performances of the PARALLEL agent for instance, the PRESET2 does ‘use silence as a structural and gestural element’. However the participant writes,

‘In the first two examples I feel this is unsuccessful as the points where the piece drops to silence feel more like endings rather than an ebb in the piece... The third example ... is more successful in its structural and gestural use of silence.’

Thus, while the agent is sometimes capable of successfully using silence as a musical element, it does not capture all of the subtle considerations involved in doing so.

The most positive comments overall were in response to the DESIGN1 and DESIGN2† agents. He/she wrote that the DESIGN1 agent

‘captured some aspects of my musical performance very accurately ... as if it was listening to the overall result and thinking about the structural
development of the piece ... I had not expected the agents to perform with such musicality’;

and of the DESIGN2 agent:

‘There is a level of musical engagement, gestural control, development and structure very similar to how [I] would approach the material ... it would be hard to separate the agents [sic] from a human performer in a listening test.’

Note that while this participant did respond very positively in general to all Likert-type questions for all agents, it was only for these two agents that he/she responded with strong agreement to all similarity and musicality statements. As mentioned above there are minor criticisms of the DESIGN1 agent relating to ‘durational and gestural aspects.’ However, for the DESIGN2 agent there were no such criticisms: ‘I would say the agents [sic] are doing nothing wrong.’

Interestingly, the lowest levels of agreement indicated by this participant were in response to question A11, regarding the variety exhibited in the agents’ performances, with respect to that exhibited in the example performances. This may indicate that while all six agents may in various ways define regions in the space of possible performances that are musically successful (at least in large part), the participant perceived these regions to be relatively small.

**Participant P3 Responses**

Like participant P2, participant P3 responded with moderate or strong agreement for all six agents, to nine of the eleven statements (see Figure 7.8). However, unlike that for participant P2, the musical style in this case clearly requires adherence to a number of strict constraints relating to hypermetrical structure (here, changes happening after multiples of four bars) and long-term structures (e.g. buildups), which certain agent types cannot capture (particularly the RANDOM agent).
7.2. Results: Participant Submissions and Responses

This participant was sent a follow-up question regarding his/her approach to answering the Likert-type questions. The response included the following:

’some errors can have large effects without really making it stylistically different’;

‘with techno u [sic] can have a weak bar that can kill the momentum, but overall its kinda [sic] going in the right direction ... similar mistakes in other genres could slip by’; and

‘it might 99% there [sic], but the 1% makes it not work. But it would seem a little harsh to rate it poorly.’

Thus, first, the participant may have interpreted the term ‘stylistically similar’ more broadly than we envisaged, and second, he/she felt that if an agent made the majority of decisions correctly, it should be rated highly, even if the small number of badly made decisions were critical.

While this provides some explanation of the participant’s approach, it may still seem insufficient to explain the high levels of agreement for the RANDOM† agent in particular. Since this agent randomly chooses clips and effects from one bar to the next, it would seem unlikely that it would be perceived to be making most decisions appropriately. However, because of the selection of variable values available to the
RANDOM agent in this case, certain basic stylistic elements were reproduced. For example, the performances did feature the constant presence of the main percussive elements. This arose because the Main Beats track was active throughout the example performances (see Section 7.2.1), thus it was active throughout the performances of the RANDOM agent (since the agent cannot generate variable values unseen in the examples). Indeed, portions of steady beats arise by chance. Another stylistic element that is reproduced by the RANDOM agent is the frequent presence of the synthesizer chords accompanied by continuous change in the cutoff frequency of the low-pass filter effect.

The 6-point Likert scale may have been somewhat too coarse for this participant to clearly differentiate between the agents in many cases. However, for the questions related to long-term structure, there is differentiation between the agents, and this is consistent with the participant’s written comments. First, he/she indicated strong agreement with the statements related to long-term structure only for the PRESET1 and DESIGN2† agents. These were the only agents for which no specific criticisms were articulated (PRESET1: ‘Good appropriateness in progressions.’, ‘Pretty solid performance’; DESIGN2: ‘Not too much wrong with these takes’, ‘Long term structure seems much more sound’), though reservations were expressed about the variety exhibited by the PRESET1 agent (‘Perhaps less variation here.’) and this is reflected in the lower level of agreement in response to question A11.

For the PARALLEL, PRESET2 and DESIGN1 agents, the written criticisms all relate to the long term structure. The comments, ‘missing climaxes and subtleties in the long term structure’, ‘Sometimes losing momentum’, and ‘Some loss of momentum’, were attributed to the three sets of performances, respectively. In addition, for the DESIGN1 agent, the participant wrote ‘Variations can offer new ideas but mistakes can kill the overall movement’.

In sum, while this participant tended towards high levels of agreement for all agents, there is a clear sensitivity to the long-term structure of the performances. In this regard the highest levels of agreement are expressed for the PRESET1 and
Figure 7.9: Responses for participant P4. The agent types are abbreviated Ra (Random), Pa (Parallel), Pr1 (Preset1), Pr2 (Preset2), D1 (Design1) and D2 (Design2).

DESIGN2 agents. Since a higher level of agreement is expressed for the DESIGN2 with regard to the level of variety exhibited by the agent, this agent might be said to have best captured the style underlying the example performances.

**Participant P4 Responses**

Participant P4 expressed a clear preference for the stylistic similarity exhibited by the DESIGN1 agent (see Figure 7.9). He/she indicated strong agreement with three of the four similarity statements and moderate agreement with the fourth. In response to question A12 (‘the main thing the agent is doing wrong’), the participant wrote: ‘nothing much!’. In stark contrast, there was no more than slight agreement with the similarity statements for any of the other agents. The written comments were consistent with this, with the other agents being criticised in general for having to many incoherent changes (RANDOM: ‘too many changes’; PARALLEL: ‘sudden changes which don’t fit’; PRESET1: ‘changing parts too much’; PRESET2: ‘changes seem random’; DESIGN2: ‘things stop and start in a way that seems random’).

The highest levels of agreement with the musicality statements are also given for the DESIGN1 agent. It is unclear why these levels of agreement are generally lower than those for the similarity statements. One possibility is that while good style similarity is generally achieved, the participant perceived certain minutiae to
indicate a lack of musicality on the part of the agent. Such minutiæ could include slightly mistimed elements or misjudged durations that, while not conspicuously inappropriate, betray a lack of musical sensitivity or awareness.

To statement A9 (concerning the potential for generating new ideas), the participant indicated strong agreement for the RANDOM agent, and various lesser levels of agreement for the other agents. Significantly, slight disagreement with this statement is indicated for the DESIGN1 agent, supporting the notion that the aim of style emulation may in some cases conflict with that of generating new ideas. The participant indicated strong agreement with statement A10 (‘mingles examples’) for the DESIGN1 agent (slight agreement or less for the others), and slight agreement with the statement A11 (more variety than in the examples) for this agent. The RANDOM agent was perceived to exhibit the most variety.

In sum, the DESIGN1 agent was judged to have achieved good style emulation, though it may have been lacking somewhat in musicality. The RANDOM agent was judged to be poor in terms of style emulation and musicality, but positively in its potential for generating new ideas. The other agents lay in between these two extremes.

**Participant P5 Responses**

Participant P5 indicated strong agreement with all similarity and musicality statements for the DESIGN1 and DESIGN2 agents (see Figure 7.10). In addition, he/she indicated moderate or strong agreement with those statements for the PARALLEL and PRESET1 agents. The relatively small perceived differences between these two pairs of agents were reflected in the written comments. Of the PARALLEL agent, the participant wrote:

‘Whilst it does keep a good long-term structure it seems to introduce clips at almost random times which do not work musically all of the time... some of its clip launches seem a bit off to my taste’;
and of the PRESET1 agent: ‘some clip launches are not perfect in terms of structure’. Similar criticisms were made of the DESIGN1 and DESIGN2 agents, though more mildly (DESIGN1: ‘It occasionally plays clips where I would not like them played if [I] was performing’; DESIGN2: ‘It seems to be performing almost identically to how I would but sometimes clips start or stop at times I would prefer them not to if I was performing.’).

For the RANDOM and PRESET2 agents, the participant indicated significantly lower levels of agreement with the similarity and musicality statements, though for different reasons in each case. Whereas the RANDOM agent’s lack of continuity was cited (‘jumping more sporadically between clips than in my examples’) the PRESET2 agent exhibited elements of continuity that were disrupted in inappropriate ways:

‘It stops clips when they shouldn’t be stopped. eg. [sic] a bass line that is building is cutoff [sic] half-way. clips [sic] are starting and stopping too sporadically. The long term structure is also affected by this as it looses [sic] its musical progression.’

The participant judged the RANDOM and PRESET1 agents most likely to generate new ideas (strong agreement with A9). This is supported in the comments (RANDOM: ‘It gave me new perspectives on some of the sounds as they were paired/sequenced
differently to how I would perform it.’; PRESET1: ‘it is coming up with structures that I like and find interesting’). In addition, all agents were judged to mingle the example performances (moderate or strong agreement with A10). The DESIGN1 agent was judged to exhibit the most variety with respect to the example performances (strong agreement with A11). Significantly, the DESIGN2 agent was judged to be among the least likely to give new ideas (slight agreement with A9) and to exhibit the least variety (slight disagreement with A11). Thus, the DESIGN1 agent may be the most successful in this case, being judged the most stylistically similar and musical but also exhibiting the most variety.

**Participant P6 Responses**

Participant P6 responded to the similarity and musicality statements with the highest levels of agreement for the DESIGN2 agent (moderate or strong agreement in each case; see Figure 7.11). Of this agent, the participant wrote

> ‘Doing much right. Some breaks seem to be too long before coming back to the main theme. Some small stutter effects seem to be a bit too much. But all in all also a very musical agent keeping the amount of variations rather limited but musical. Also the long structure seems to have some logic and makes sense.’

While the other agents were positively regarded in some respects, all were significantly flawed: in each case, the participant indicated either moderate or strong disagreement with at least one of the similarity or musicality statements. The participant’s written comments cited the lack of a ‘system behind the variations’ for the RANDOM agent; the overuse of variation (‘too much variation, sounds a bit random’) for the PARALLEL agent; and the lack of long-term structure, ‘too much variation at the same time on different elements’, and the presence of musical elements which ‘are not in harmony’ for the PRESET1 agent. For the PRESET2 and DESIGN1 agents, the flaws were not outlined in detail.
The participant judged all agents to have potential for generating new ideas (moderate or strong agreement with A9), and that all agents mingled the examples (moderate or strong agreement with A10). Only the PARALLEL agent was judged to exhibit significantly less variety than in the example performances (moderate disagreement with A11; for all other agents there was moderate or strong agreement).

**Participant P7 Responses**

Participant P7 responded to the similarity and musicality statements with the highest levels of agreement for the DESIGN2† agent (moderate or strong agreement in each case; see Figure 7.12). He/she wrote that the agent was

‘Extrapolating the material and providing a good balance between vertical and long term structure. Nothing too abrupt, yet the layering is not predictable either... I much prefer the performances of this agent than any of my 4 example performances! Really subtle and considered arrangement of material, and a great long-term structure! The momentary clips (foley sounds) were often used repeatedly which was an interesting variation on the way I worked with these in the examples.’
This is consistent with the participant’s strong agreement with all musicality statements, whereas there was only moderate agreement with three of the four similarity statements (and strong agreement with the fourth).

Also noteworthy is the participant’s moderate agreement that the RANDOM agent exhibited short-term and vertical musicality. The participant seemed excited by the unusual performances of this agent: ‘This performance has given me a completely different conception for how the material I created for use in performance might be used in a different way.’

The comments regarding the remaining agents were largely focused on the temporal aspects of the layering of sound materials. Of the PARALLEL agent, the participant wrote:

‘A few abrupt stops and starts. The structuring of the layering sounds a little out of place ... the build up of materials was at times a little abrupt, with not much space for progression from one state to another.’;

of the PRESET1† agent: ‘The texture is too sparse in general ... One sound objects [sic] is held for an extended amount of time without a lot of “evolution” in the structuring of materials’; of the PRESET2 agent: ‘The changes between material are less layered and evolving, and more abrupt.’; and finally, of the DESIGN1 agent: ‘there are certain
combinations of materials that the agent chooses to prolong for longer than was exhibited in my examples.’

The participant moderately or strongly agreed that four of the agents could generate new ideas for using the musical material (the PARALLEL and PRESET2 agents were the exceptions). In addition, he/she agreed at least slightly that the agents mingled the example performances in all cases. Finally, the participant strongly agreed that the RANDOM agent exhibited more variety than in the example performances. For the DESIGN2 agent, he/she only slightly agreed, thus this agent, in light of the responses to the similarity and musicality statements, may define a portion of the space of possible performances that is close to, and perhaps approximately the same size as that spanned by the example performances.

**Summary of Participant Responses**

For each agent and each Likert-type question, the median answer across participants is shown in Figure 7.13, and a histogram of the response frequencies across participants is shown in Figure 7.14. Broadly, for the similarity and musicality questions, the lowest levels of agreement are associated with the RANDOM agent, and the highest, with the DESIGN agents. Intermediate levels of agreement are associated with the PARALLEL, PRESET1 and PRESET2 agent types. This trend is not apparent—and perhaps it is even reversed—for those questions related to the potential for agents to generate new ideas (A9) and the extents to which they are judged to mingle the example performances (A10), and exhibit more variety than the example performances (A11). In the following, we discuss this in detail.

The RANDOM agent type was generally judged to be poorest in terms of style similarity and musicality, and particularly the long-term aspects of these. Interestingly, there are the relatively high levels of agreement with the vertical musicality of the RANDOM agent (four participants moderately or strongly agreed with statement A8). However for the compositions submitted by most participants, there were few combinations of material that were conspicuously inappropriate (e.g. all remained
in the same key throughout). In addition, since the RANDOM agents made random selections of new variable values at every decision point, inappropriate combinations were unlikely to remain for long. Concerning this, participant P7 commented ‘With respect to style similarity, it sounds like the vertical structure makes sense, but it’s hard to tell because of the rapid shifts in material.’

The RANDOM agent type was judged most positively in its potential for generating new ideas (median response to A9: strongly agree), though there were indications (see, e.g., the comments of participant P1 above) that the ideas generated were generally for novel short sequences or momentary juxtapositions of material. In addition, the RANDOM agent type was judged to exhibit the most variety in its performances (median response to A11: strongly agree). This is interesting, since RANDOM agents characteristically exhibit a great variety of momentary ideas but very little variety in long-term structure. Clearly, the former had a greater impact on participants’ responses.

The responses to the similarity and musicality statements are quite mixed for the PARALLEL, PRESET1 and PRESET2 agent types, though interestingly, for the statements related to overall style and musicality (A1, A2), the responses were generally higher than those for the statements related to individual aspects. The
Figure 7.14: Histograms of response frequencies across participants for each agent type and each question.
broad distributions of responses may indicate that the effectiveness of these types of models depends strongly on the specific musical style and material involved. For the \textit{Preset1} and \textit{Preset2} agents there is a slight indication of a bimodal distribution for some questions (see particularly \textit{Preset1}, questions A1 and A3; and \textit{Preset2}, questions A1 and A4, Figure 7.14). This may be evidence that these learning configurations might work quite well in some cases while in others, they lead to significant behavioural flaws. Finally, as with the \textit{Random} agent type, the \textit{Parallel}, \textit{Preset1} and \textit{Preset2} agent types were generally judged positively in their potential for generating new ideas (moderate or strong agreement with A9). The responses related to the variety exhibited by these agent types are overall more equivocal.

For the \textit{Design1} and \textit{Design2} agent types, the responses to the similarity and musicality statements show the highest levels of agreement, with \textit{Design2} being judged more positively. Broadly, this indicates that an ‘expert’ user of the ADTK can design agents exhibiting higher levels of musicality, and better able to emulate specific styles, than any of the pre-determined learning configurations used in this study. That the \textit{Design2} agent type was generally judged more positively might be expected, since the sequential way in which agents were created and used to generate performances allowed the flaws in the performances of the \textit{Design1} agent to inform the creation of the \textit{Design2} agent (though note that the procedure outlined in Section 7.1.4 for generating example performances was strictly adhered to).

The \textit{Design1} and \textit{Design2} agent types were judged less favourably, in general, in their ability to generate new ideas. However, as previously noted the aim of generating new ideas may conflict with that of style emulation in many cases. Similarly, the aim of style emulation, which was the focus of the \textit{Design} agents, perhaps leads to more highly-constrained designs and accounts for the lower levels of agreement that these agents exhibited more variety than was exhibited in the example performances (however, we note that no \textit{Design} agent was created such that it would simply copy the example performances; in every case, the generated performances were significantly different from each individual example performance, and the responses
to question A11 support this).

Overall, these results reflect the greater level of sophistication possible with models that utilise the rule learning and especially the feature-selection options in the Agent Designer. In addition, there is evidence to support the notion that the PRESET designs might be useful in some cases, but they are far from a ‘one-size-fits-all’ solution to the creation of an effective learning configuration. The RANDOM and PARALLEL agent types were included largely to provide a baseline from which to judge the other types. However, they too may have uses, particularly with respect to the generation of new ideas.

To finish, we note the broad range of approaches to answering the Likert-type questions, with certain participants tending toward positive responses. Generally, the responses were consistent with the written comments, however, and in each case, it was clear which agent or agents were judged by the participant to have achieved greatest style similarity and exhibited the highest level of musicality. While this is useful, it does not provide a clear indication of how close these agents are to being usable in practise. In order to address this, a post-hoc step was taken in which each participant was asked to directly assess their confidence that the best agent (i.e. the one of the six that he/she preferred) would produce a satisfactory performance in a live performance setting. This is reported in the following.

### 7.2.3 Direct Assessment of Agent ‘Usability’

After completing the feedback questionnaire, all participants were sent a follow-up question which read:

The following question assumes that the composition you used for the study was ready for public performance and the examples you submitted were representative of the way you’d like to perform it live. Even if these things were not entirely true, please answer as if they were.

**Question:** If you were in the middle of a live performance and for some
Chapter 7. Modelling Arrangement-Level Musical Decision Making with the ADTK

<table>
<thead>
<tr>
<th>0%</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Confidence</td>
<td>Moderate Confidence</td>
<td>Complete Confidence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.11: The percentage confidence scale presented to participants. The alignment between labels and numbers is as it was presented, with ‘No Confidence’ left-aligned with 0% and extending almost to 20; ‘Moderate Confidence’ left-aligned with 40 and extending to just after 60; and ‘Complete Confidence’ left-aligned with 90.

reason, you needed an agent to play the composition instead of performing it yourself (perhaps you need to attend to a technical issue), how would you rate your confidence that the agent you think is best overall would do a satisfactory performance in your place?

To guide their responses, participants were presented with a percentage scale as shown in Table 7.11, though they were asked to respond with a numerical figure, rather than by marking the scale.

The responses to the post-hoc question are shown in Figure 7.15. They range between 60% and 95% with a mean of 76.4% and standard deviation, 13.8%. Participants P4 and P6 reported the joint-lowest confidence. Participant P5 reported the highest confidence that the agent’s performance would be satisfactory. Participant P6, who reported 60% confidence, qualified his/her answer as follows:

‘If the track has no real climax that has to be built up slowly but a flowing arrangement with some random variations, the agent can do good work. Regarding the situation, in a proper performer situation where the main goal is to adapt and react to the audience and time the parts and climax right, I don’t think the agent can work properly. In a situation where music is in the background and does not have to adapt to external factors, it would work.’

Figure 7.16 shows normalised plots of each participant’s confidence alongside his/her mean normalised response to the Likert-type questions in various groupings.
Specifically, the means across all Likert-type questions are shown, along with that across those related to style similarity (A1, A3, A5 and A7) and those related to musicality (A2, A4, A6 and A8). Pearson’s correlation coefficient was calculated between each of the sets of values for the Likert-type responses, and the responses to the post-hoc question. The highest correlation was between the responses to the musicality statements and the responses to the post-hoc question (0.73). The same calculation was performed for each Likert-type question separately, and the highest correlation (0.76) resulted jointly for statement A4 (long-term musicality) and A8 (vertical musicality). This provides an indication, albeit from a small number of participants, that judgements of musicality are more important than those of style similarity when assessing an agent’s readiness for live performance.

### 7.2.4 Uses of ADTK Agents

Participants’ responses to the four final Likert-type questions (F1-F4; see Table 7.3) are shown in Figure 7.17. Six of the seven participants strongly agreed that the ADTK could be used to create systems that generate ambient music and systems that generate sound for art installations. For the other two types of music, the responses were more mixed. The median response to statement F3 (use in classical music) was
Figure 7.16: Correspondences between post-hoc question responses and Likert-type responses. For each participant, the following values are shown, all normalised to between 0.0 and 1.0: (i) the response to the post-hoc question, (ii) the mean response across all Likert-type questions, (iii) the mean across all Likert-type questions related to style similarity (A1, A3, A5 and A7) and (iv) the mean response across all Likert-type questions related to musicality (A2, A4, A6 and A8).
‘slightly disagree’, and that to statement F4 (pop music) was ‘slightly agree’, with a broad spread of responses in each case.

Participants suggested other uses for ADTK agents both in response to question F5 (again, see Table 7.3) and occasionally in their comments for individual agents. Four of the seven participants (P1, P2, P4 and P7) suggested the use of the ADTK for use in music composition as a tool for generating new ideas. For example, participant P2 suggested that it could be used for ‘opening up new possibilities and providing a fresh look on material’; P4, that it could be used for ‘generating possible ideas when sequencing out an arrangement in electronic music; and P7 suggested its ‘use as an ideas generator during the compositional process of an electro-acoustic work’.

Three participants (P1, P5 and P6) suggested that ADTK agents, despite being non-interactive, could perform alongside a musician in a live performance context. In particular, P1 suggested its use ‘as an “interactive” backing track in a live set,’ while P5 wrote:

‘If I were performing with Ableton whilst also using other musical equipment (e.g. analogue synthesizers, sequencers or any other instruments) an agent could be used to perform certain aspects of a performance allowing me or anyone else performing the freedom to focus on other instruments.’

Participant P6 suggested that ADTK agents could be employed in individual Ableton Live tracks ‘manipulating specific parameters on these specific tracks.’
Finally, a number of speculative suggestions were made that could provide avenues for future research. These included the use of ADTK agents for creating ‘visual output using software-based systems,’ and for generating ‘static visual imagery based on a users input’ (P2); for ‘track sequencing in DJing [sic]’ and ‘use at note level ... so that internal aspects of the clips ... could be varied’ (P3); and as ‘a beat programming helper’, or, ‘maybe for visuals’ (P6).

### 7.3 Results: Agent Design Techniques

We use the term ‘design technique’ to refer to a way of augmenting or modifying a learning configuration to capture a specific musical pattern. In general, it is difficult to rigorously infer a causal connection between a specific design technique and specific participant responses. However, in the following, we discuss a range of techniques, particularly ones that were used in agents which garnered positive responses with regard to style similarity and musicality. These techniques are potentially widely applicable, and can inform the development of high-level user interfaces for the Agent Designer.

#### Count of the Number of Tracks Playing

To model overall dynamic structure, a count of the number of tracks playing can be modelled with a VMM. This method formed part of the rationale for the inclusion of custom variables in the ADTK. A custom variable is created representing a count of the number of Ableton Live tracks playing, or alternatively, the number of active sound sources in a Max patch. As originally described (see Equation 5.16), it was used for the P1-DESIGN1 agent⁷, as well as in all of the PRESET2 agents. However, this technique can be problematic. For example, consider a situation in which the number of active tracks remains constant for a period of time at a value of four. Four tracks must be active for this entire period, but the selection of tracks may change.

---

⁷Throughout the following, we will use this compact way of referring to a particular agent: Participant-Type.
Figure 7.18: Inappropriate interactions caused by modelling the number of active tracks. The priorities of the tracks are shown on the left-hand side (lower numbers are higher priorities). At the bar marked ‘A’ the, the four low priority tracks (those with priorities 8, 9, 10 and 11) are forced to stop because four higher-priority tracks begin to play. At ‘B’ the priority-1 track starts and forces the priority-11 track to stop, and at ‘C’ the priority-3 track starts and forces the priority-10 track to stop.

Thus, since the variables are considered in priority order, this means that the highest priority track can determine the activity of the lower priority tracks: when it turns off, one of the lower priority tracks may be forced to turn on (to maintain a total of four) and vice versa, and in some cases this can be stylistically inappropriate. Examples can be seen in Figure 7.18 which shows a portion of one of the performances by the P5-PRESET2 agent\(^8\). Comments made by participant P5 regarding this agent are consistent with these interactions between tracks (‘It stops clips when they shouldn’t be stopped’).

In a number of agent designs, this problem was alleviated somewhat by coarsely quantizing the count of the number of tracks playing. For example, if the custom variable representing a count of the number of tracks is denoted \(c_1\), then a second custom variable, \(c_2\) can be created as follows:

\[
c_2 = \left[ \Sigma (>\text{ANY} (0, c_1), >\text{ANY} (2, c_1), >\text{ANY} (4, c_1)) \right]. \tag{7.1}
\]

That is, \(c_2\) will have a value of 0 if there are no tracks playing; 1, if there are 1-2 tracks playing; 2 if there are 3-4 tracks playing; and 3 if there are more than 4 tracks playing.\(^8\)

\(^8\)This recording is available on the webpage accompanying this chapter.
playing. Since $c_2$ is a coarse approximation of the number of tracks playing, it can be used in place of $c_1$ to roughly model the overall dynamic structure while reducing the number of undesirable interactions between individual track variables. This technique was used in agents P4-DESIGN1 which was the agent judged by participant P4 to be the most stylistically similar agent.

Techniques for further fine-tuning this technique include modelling the count variable ($c_1$ or $c_2$, above) as a BLOCK custom variable so that changes are consistent with the hypermetrical structure (insofar as they are in the training data set). This was used in P5-DESIGN1 and P5-DESIGN2, both of which were judged much more positively than the P5-PRESET2 agent that produced the performance shown in Figure 7.18. Additionally, the lower priority track variables can be given equal priority so that values are chosen for them at each decision point in a random order. This can reduce the conspicuousness of stylistically inappropriate interactions.

Finally, we note that simply including a custom variable to represent the number of tracks playing (even without a VMM) is sufficient to place bounds on the number of tracks that play at any one time. For example if there is always at least one, and not more than four tracks playing in the training data set, then the custom variable will have the domain $\{1, \ldots, 4\}$ and the agent’s decisions will be restricted correspondingly. This technique was used in the P7-DESIGN1 agent.

**Count of the Number of Tracks that Turn On and Off**

A second technique for modelling overall dynamic structure involves creating a custom variable which is a COMBO of (i) a count of the number of tracks that turn on and (ii) a count of the number of tracks that turn off, at the current decision point (see Figure 7.19). By applying a high-order VMM to this variable, it is possible to effectively capture the style in which musical material is introduced and removed in the training data set. This technique was used in agents P2-DESIGN1 and P7-DESIGN2. Of the latter, participant P7 commented ‘Really subtle and considered arrangement of material, and a great long-term structure!’
7.3. Results: Agent Design Techniques

Figure 7.19: Modelling how tracks or instruments are layered. Shown is some training data in which four tracks are started and stopped at various times. Two custom variables are created to count (i) the number of tracks turned on, and (ii) the number of tracks turned off at each decision point. A VMM is then used to model a Combo variable representing the tuple of these two custom variables. In this example, a generated performance would never feature two tracks being turned on at once, since this never happens in the training data set.

**Restricting Track ‘Interchangeability’**

The first two techniques above essentially involve defining a set of tracks that are musically equivalent in some respect. By modelling the number of tracks, or the introduction and removal of tracks in a way that is separate from the precise selection of tracks, the tracks become interchangeable in the model. While this may be broadly true in many cases (e.g. the instruments are gradually layered on top of one another, but the precise ordering does not matter), there is often a need to restrict the model in some ways. For example, if the number of tracks playing is modelled and in the training data set the performances begin with a single track playing, it is unlikely that it is appropriate to use any track at all. Informally, we found that when only a small number of tracks is playing, the precise selection of tracks is often more important than when a large number of tracks is playing. The techniques above can be adapted for specific musical material by including a rule group containing the custom variable representing the count of the number of tracks, and the custom variables representing the on/off state of each of the individual tracks. Thus, rules can be found such as

\[
\text{# Tracks} = 1 \implies \text{Hi-Hat On} = 0,
\]

meaning ‘if there is only one track playing, it should not be the hi-hat’ or equivalently, ‘the hi-hat should never play by itself.’ This technique was used in agents P4-
Figure 7.20: Structure imposed by rules. Shown is a portion of a performance by the P6-Preset1 agent. Despite no custom variables being used, some structure is imposed by the rules: at ‘A’ the ‘yellow’ clip (in a high-priority track) forces two other clips to begin playing at the same time.

DESIGN1, and P5-DESIGN1.

**Identify Structurally Important Track(s)**

A final technique for modelling overall dynamic structure involves identifying a small number of tracks (or perhaps just one) that broadly determine the activity of the remaining tracks through rules and other features of the model (such as the techniques for capturing short-term stylistically salient patterns presented below). Once such a set of tracks has been identified, they are modelled by a high-order VMM applied to a single COMBO variable to which a high priority is attributed (a COMBO variable is only required if there is more than one ‘important’ track). Figure 7.20 shows an example of a performance of the P6-TECH1 agent in which coordination between tracks is imposed by the discovered rules. This technique was used in the P3-DESIGN1, P6-DESIGN1 and P6-DESIGN2 agents, as well as in all of the PRESET1 agents. However, its efficacy is extremely dependent on the specific training data set since it requires that dependencies can be found to impose sufficient structure.
Simultaneous Changes

In a number of training data sets there were clear patterns in which changes in one variable coincided with changes in another. Broadly there are four types of patterns:

• Any change in one variable causes any change in another.
• Any change in one variable causes a particular change in another.
• A particular change in one variable causes any change in another.
• A particular change in one variable causes a particular change in another.

Of course, more complex patterns can arise among groups of three or more variables.

These patterns can be modelled by using previous custom variables to represent the value of a particular variable. An all equal custom variable can then be used to indicate if any change has occurred. For example, in the following equation, the custom variable, $c_3$ has a value of one if the variable $p_1$ is equal to its previous value:

$$c_3 = \left[ \succ (p_1, \text{PREV}(p_1)) \right]. \quad (7.2)$$

Specific changes can also be found, for example, the following equation defines a custom variable, $c_4$ that has the value 1 when another variable ($p_1$) has changed from three to four, and zero otherwise:

$$c_4 = \left[ \prec (1, \succ (3, p_1), \succ (4, \text{PREV}(p_1))) \right]. \quad (7.3)$$

Once the appropriate custom variables have been created, they can be added to rule groups to discover rules corresponding to the pattern types listed above. This technique was used in the P6-DESIGN1 and P6-DESIGN2 agents.

Defining Equivalence Classes

Finally, we note that one of the uses of custom variables that formed part of the rationale for the Agent Designer was to define ‘equivalence classes’ or sets of variable
values that are interchangeable (see Section 5.4.6). This technique was used in, for example, the P1-DESIGN1 and P1-DESIGN2 to model the fact that three of the clips on the Bass track were interchangeable, being variations on a single musical line. It was also used in the P6-DESIGN2 agent to model the equivalence between a set of different effects that could be turned on as a musical fill, to delineate the end of a four-bar phrase.

7.4 Discussion

In this section, we discuss the above study from three perspectives. First, we broadly characterise the modelling capabilities of ADTK agents, and suggest how some of their shortcomings might be addressed. Second, we consider the results with regard to the extent to which they can inform the design of our proposed high-level user interface for the Agent Designer. Finally, we consider the possible sources of error in this study, their potential ramifications, and their relation to future work.

7.4.1 The Modelling Capabilities of ADTK Agents

Four participants indicated that they were at least 80% confident that their preferred agents would perform satisfactorily in a live context. Each of these participants (P2, P3, P5, P7) submitted compositions in which the layering and juxtaposing of patterns was used to create variations on a musical structure which was either quite tightly constrained, in the cases of P3 and P5 (techno), or quite open in the cases of P2 and P7 (textural and ambient).

Broadly, the compositions submitted by the three remaining participants were different in that they had

- distinct sections, that were

- characterised by a number of features, perhaps not well-defined, involving multiple tracks.
Specifically, participant P1 submitted a composition with distinct sections, characterised by multiple attributes including the choice of percussion; P4 submitted a composition in which instruments and patterns were rapidly combined and recombined to create buildups and breakdowns in a highly articulated manner; and P6 submitted a composition involving sections clearly demarcated by abrupt (but variable) changes in instrumentation. The composition submitted by participant P3 also had distinct sections (introduction, buildup, etc.) however they were largely determined by the activity of a single Ableton Live track (the MainBeats). Participants P1, P3 and P4 had significantly lower confidence that their preferred agents would perform satisfactorily in a live context (though P6 qualified his/her answer indicating that for ‘background’ music, rather than live performance, the agent might be quite satisfactory). Thus, the results of this study support the position that the ADTK can be used to design and implement agents that effectively model what might be loosely termed ‘single-section’ compositions and in some cases, it can be used for agents that convincingly capture the higher-level structure of more complex compositions, as was done to some extent in the cases of participants P1 and P6, and less so for P4. This corresponds well to the fact that the ADTK was designed with arrangement-level musical decision making in mind, and not structure on the macro time scale.

In the following, we propose a number of ways to improve the modelling capabilities of ADTK agents. Most important, we address the issue of capturing distinct sections in a composition, i.e. modelling musical structure on the macro time scale. In addition, we note two other issues which, while less significant, also reduce the modelling capabilities of ADTK agents. The first is that it is not possible to break the hypermetrical structure (i.e. the global temporal alignment of BLOCK variables) even when it may not be required throughout a performance. The second is the possibility that long periods without change will arise, as can happen by chance using Markov models.
Modelling High-Level Structure

To begin, consider the pattern of values shown in Figure 7.21. In this pattern, the statistics of variable $p_2$ depend on the value of variable $p_1$. That is, when $p_1$ is orange, $p_2$ is green more often than when $p_1$ is blue. This is a simple example of a structure with distinct sections that are not characterised by the activity in a single track. The only way to model this in the ADTK is by creating a COMBO custom variable to treat $p_1$ and $p_2$ as a single variable. However, this is not straightforward when, for instance, more than two variables are involved, since creating a COMBO variable to combine numerous other variables, tends to lead to overfitting (only combinations found in the training data set can be used). A more satisfactory way to model the pattern in Figure 7.21 would be to train two VMMs to model $p_2$, one that would be used when $p_1$ is blue, and the other, when $p_1$ is orange.

One way to address the issue of modelling separate sections would be to explicitly add a higher level of decision making to ADTK agents and a straightforward way to do this would be to allow a single learning configuration to be trained separately on different parts of the training data set. In an additional design step, the user could mark portions of the training examples according to their high-level musical functions (as ‘buildups’, ‘breakdowns’ or ‘sustained, high-intensity’ sections, for example). Separate sets of VMMs and rules could be trained on these portions and then a higher-level decision making process could switch between sets of VMMs at performance time.

Supporting a higher level of organisation in this way is reminiscent of the real-time switching between ‘oracles’ (i.e. statistical models) that is possible with the Omax autonomous improvisation software [15]. In addition, the explicit inclusion
of this level of organisation higher than the ‘pattern’ is also a feature of systems for generating electronic dance music such as GEDMAS [10] and GESMI [76]. We do not envisage that it would add significant complexity to using the ADTK, since it does not require extra agents to be designed (though this could be offered as an option), but simply markers to be placed in the training data set. In fact, we speculate that in many cases simpler agent designs would suffice.

**Breaks in Hypermetrical Structure**

While certain musical styles generally have strict hypermetrical structure, individual performances may begin with introductory sections or include breakdown sections that do not adhere to this. This was an issue, for instance when designing agents for participant P6: though there was a strict four-bar hypermetrical structure, there were introductory sections of for instance six bars, that did not fit in. Thus the modelling might be made more flexible by attaching the structure to certain instruments. Specifically, the decision point number (which determines the position in each block, and thus, when new blocks begin) might only be specified when particular instruments are active (for instance the main percussion instruments). This would permit introductions and breakdowns of odd lengths while maintaining hypermetrical structure when it is required.

**Sensitivity to Duration**

A number of participants commented on prolonged durations during agent performances during which there was not enough change (P2 and P7, in particular). As has been discussed previously, long durations are difficult to model with Markov models, without overfitting. It may be useful to investigate the inclusion of options that would allow durational bounds to be learned from the training data set, so that patterns such as ‘never proceed for more than ten bars without a change’ could be discovered. The implementation details of this modelling enhancement, along with those proposed above, are discussed in Section 10.3.1.
7.4.2 Informing the Design of a Higher-Level User Interface

We can use the results presented in the previous sections to inform the design of a high-level user interface for the Agent Designer. We envisage a multi-layered design (see, e.g. [159]) with different layers corresponding to different levels of user expertise. For the purposes of this discussion, we will consider such an interface comprising three layers:

- **Layer 1** in which the user can simply select between preset learning configurations, some of which might require additional information, such as identification of important tracks or hypermetrical structures.

- **Layer 2** in which the user could choose subsets of variables and apply design techniques in order to, for example, ‘model the number of these tracks that are playing’ or ‘capture the good combinations of these effects.’

- **Layer 3** which might be similar to the current user interface of the Agent Designer (see Appendix A).

The results of the study reported in this chapter can inform the design of this interface in a number of ways. First, the characterisation of the four pre-determined learning configurations can help determine whether they might be fruitfully added as options to Layer 1, what modifications might improve them, and also, what additional presets might be required. Second, the agent design techniques identified in Section 7.3 can form the basis of the set of modelling techniques available in Layer 2. Finally, a combination of the agent design techniques, and the characterisation of the pre-determined learning configurations can be used as the basis for the development of new preset learning configurations that might themselves be characterised in another iteration of this study. In the following, we discuss each of these topics in turn.
7.4. Discussion

**The Preset Learning Configurations**

**RANDOM**: The results indicated that the RANDOM learning configuration would be useful, primarily for generating ideas for novel ways of using musical materials. While its functionality can be readily duplicated in Max or Ableton Live, its inclusion in Layer 1 would make comparisons with other presets straightforward. In addition, it would provide a basic starting point from which to develop more sophisticated agents (i.e. choose the preset in Layer 1 and then modify it in Layer 2).

**PARALLEL**: Participant responses for the PARALLEL configuration were quite mixed. While this configuration is capable of capturing patterns in the sequences of individual variables, it cannot capture dependencies between variables. These characteristics were reflected, for instance, in the comments made by participant P2 on the P2-PARALLEL agent, specifically that it produced ‘gestures ... similar in style and substance’ (i.e. capturing patterns in individual variable values) but also that it generated combinations of parameter values unlike those in the example performances (i.e. dependencies between variables were not captured). This potential for PARALLEL-type agents to arrive at previously unseen combinations of parameter values gives it potential to generate new ideas. However, among the unseen combinations that can arise there may be catastrophic ones, such as turning all instruments off, which may continue for extended periods. It might be more effective to include a modified version of the PARALLEL configuration in which, for instance, there is a custom variable to count the number of tracks playing (or the number of instruments). This would serve to place bounds on the number of tracks playing (as described in Section 7.3) and therefore prevent all tracks being off at once (insofar as this is the case in the example performances). Similar augmentations could place bounds on the number of active effects, for example.

**PRESET1**: The PRESET1 configuration represents a simple augmentation of the PARALLEL configuration in which rule learning is included to discover dependencies between the variables. Arguably, agents created with this learning configuration were the most highly rated of the four pre-determined types (the median response
values are generally higher than those of the other three). We showed in the last section that the rules can add structure to performances by, for example, causing certain variables to change simultaneously. The specific rule-learning parameters used in the PRESET1 learning configuration could in theory be problematic since requiring less than 100% confidence can result in conflicting rules being discovered such that there are no sets of compatible variable values (this is discussed in Section 5.4.4). However, it would be straightforward to detect this and raise the required confidence automatically (until a minimum number of allowed configurations is reached, for example). Finally, we envisage that the PRESET1 learning configuration would also provide a useful starting point from which to develop more sophisticated agents.

**PRESET2:** The PRESET2 learning configuration was the most complex predetermined configuration used in the study reported here. It includes two important elements for capturing musical patterns:

- a custom SUM variable to model overall dynamic structure by counting the number of active tracks or instruments, and
- custom BLOCK variables to model hypermetrical structure.

However, as noted above, modelling dynamic structure in this way can be problematic (Section 7.3). Various ways of mitigating these problems were proposed, but their efficacy will in general depend on the specific characteristics of the training data set. Further study of this preset is required, perhaps to investigate its potential as the basis of a ‘wizard’-type interface in which the user is asked a series of questions relating to hypermetrical structure and instrument layering in order to create an effective agent.

**Design Techniques for Layer 2**

A mock-up of an intermediate-level interface for the Agent Designer is shown in Figure 7.22. Included are the various techniques from Section 7.3 for modelling
overall dynamic structure. Thus for example # Tracks Playing corresponds to an automated procedure that would set up the necessary custom variables to model the number of tracks playing. Similarly, Track Layering corresponds to another procedure that would set up the necessary custom variables to model the numbers of tracks turning on and off at each decision point (see Section 7.3). Also included are options such as those to bound the number of tracks playing or the number of effects turned on. The underlying effects of these would be similar to the # Tracks Playing option but a VMM would not be applied, thus the numbers of tracks or effects would simply be bounded to the ranges observed in the training data set.

The interface might also allow blocks (i.e. BLOCK custom variables) and VMM orders to be defined for each individual variable, as well as groups of variables among which dependencies will be sought, both between values and changes (again, see Section 7.3). Finally the interface might allow the definition of ‘equivalence classes’, meaning sets of variable values that are interchangeable (these can be achieved with custom variables as described in Section 7.3). The presentation of these features, and particularly the related terminology and documentation will be addressed in future research.

**Additional Presets**

As noted in Section 7.2.2, it is clear that none of the preset learning configurations studied were universally effective. However, our aim is to provide a set of presets such that for a wide range of musical styles and material, there are presets that, even if not ideal, would provide a good starting point for customisation. At least three of the four preset learning configurations, with the modifications suggested above, would provide an initial selection.

To arrive at new presets, we propose further iterations of this study, in which the findings from each iteration inform the presets tested in the next. For example, the changes suggested above for the PARALLEL learning configuration would be implemented before using it in a second iteration. In addition, a new preset learn-
### Learning Options

<table>
<thead>
<tr>
<th>Dynamic Structure</th>
<th>Block Size</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td># Tracks Playing</td>
<td>Choose #</td>
<td>8</td>
</tr>
<tr>
<td># Tracks Quantized</td>
<td>Choose</td>
<td></td>
</tr>
<tr>
<td>Track Layering</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Important Tracks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Individual Variables</th>
<th>Block Size</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track 1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Track 2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Track 3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Effect 1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Effect 2</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bounds</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Count # Tracks Playing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Count # Effects Turned On</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Simultaneous Values**
- Group 1: Tracks
- Group 2: Effects

**Simultaneous Changes**
- Group 3: Tracks and Effects

**Define Equivalence Classes**
- Class 1: Drum Fills
- Class 2: Bass Variations

Figure 7.22: Mock-up of Layer 2 of an Interface for the Agent Designer.
ing configuration could be created, based on the layering technique described in Section 7.3 that was very effectively used in P8-DESIGN2. Other new preset configurations might include carefully constructed searches for rules describing simultaneous changes between parameters.

### 7.4.3 Possible Sources of Error

In this section, we discuss the potential sources of error in this study and their relation to future research on the ADTK. To begin, we suggest that the three most significant potential sources of error were,

- that the experimenter did not have a veridical understanding of participants’ musical concerns in all cases;
- that the experimenter did not achieve optimal designs in all cases; and
- that the participants were not representative of users more generally, either in the musical material they submitted, their performances or their judgements of agent performances.

It is possible that the experimenter’s understanding of participants’ musical concerns was imperfect in some cases. However, where such issues arose, they would have been most likely to negatively affect the agent designs and thus the responses of the participants’ in question. Thus, the results obtained for the DESIGN agents may be seen as a baseline characterisation of the affordances and limitations of the ADTK from the point of view of an expert user. Future studies will be primarily concerned with participants’ own use of the software, in which there will be no issue of veridical understanding on the part of the agent designer.

That the experimenter did not achieve optimal designs in all cases is almost certainly true. For example, one of the criticisms made by participant P6 of the DESIGN2 agent was the overuse of ‘small stutter effects’. In another design iteration (‘DESIGN3’) this could most likely be dealt with by adding rule groups to discover situations in which it is inappropriate to use the stutter effect: it is provably possible
to find such dependencies since with appropriately configured rule groups, rules can be found that will restrict parameter combinations only to those used in the training data set; see Section 5.4.4. Irrespective of this example, suboptimal designs would also serve only to depress the results obtained for the DES IgN agents. However, unlike the issue related to veridical understanding mentioned above, it cannot be assumed that users of the software will in general be able to achieve optimal designs. The question of how the software might make it easier to achieve optimal designs is addressed in the next three chapters and it is central to our future work. In the context of this study, the possibility that more effective designs existed reinforces the viewpoint that only a baseline characterisation of the DES IgN agents has been carried out.

Finally, there is a possibility that the seven participants in this study did not provide an good representation of users more generally. It would certainly be beneficial to include a greater number. However, despite the prevalence of the terms ‘techno’ and ‘ambient’ in participants’ characterisations of their musical material (each term arose three times), there was a reasonably wide range, including abstract electroacoustic textures (P2 and P7), highly structured, beat-driven music (P3 and P4), and slower-tempo electronic and ambient music (P1, P5 and P6). There is still the possibility that participants’ judgements in relation to their own musical material would differ from the judgements that other musicians might make. In particular there is a possibility that agent performances that closely, but imperfectly, reflected a participant’s style might have been judged somewhat harshly, while agents that offered surprising and novel performances with no relation to the demonstrated style (e.g. the RANDOM agents) might have been judged sympathetically. While we are primarily interested in the extent to which the ADTK would be useful to musicians in their own practise, it would be interesting to recruit additional musicians to the study in order to compare their judgements of the seven sets examples and agent performances, to those of the participants that created the material.
7.5 Conclusion

The results reported here substantiate the claim that the ADTK is a widely applicable tool for designing musical agents (see research question III-(iii), Section 1.6.2). We have characterised its modelling capabilities and identified a number of improvements that can be made. In particular, we have identified the need for modelling structure explicitly at the level of musical sections (i.e. on the macro time scale) and proposed a straightforward way of extending the ADTK to accommodate this.

Two strategies were explored for developing a high-level user interface for the ADTK. The first involves the use of simple pre-determined learning configurations that might be selected by a user. Four such ‘presets’ were characterised and recommendations were made for their improvement. Moreover, an iterative method was proposed whereby the study could be repeated with each repetition being informed by the previous one, in order to arrive at a more complete selection of presets. The second strategy is to provide a set of modular ‘building-blocks’ from which selections might be made to form complete learning configurations. To this end, we identified six effective agent design techniques and proposed their inclusion in an intermediate layer of a multi-layer user interface.

This study did not encompass the use of ADTK agents in interactive settings. We envisaged that a requirement to identify materials, create example performances and evaluate agents for interactive contexts would have significantly reduced the number of potential participants. In the next chapter, we begin to explore the development and use of ADTK agents for interactive performance contexts.
Chapter 8

Case Studies: Agents Designed

8.1 Methodology ......................................................... 275

8.2 Case Studies ......................................................... 277
  8.2.1 Automato .......................................................... 277
  8.2.2 Many Piano Chops .............................................. 287
  8.2.3 Isidores ............................................................ 293
  8.2.4 Cello Improvisation System .................................... 299
  8.2.5 Backgammon ...................................................... 308

8.3 Discussion .......................................................... 316
  8.3.1 The Use of the ADTK in Bricoleur-Style System Development ........................................ 316
  8.3.2 The Agent Design Workflow .................................... 318
  8.3.3 Designing Interactive Agents .................................... 320
  8.3.4 Reflections on the Agent Design Process ......................... 322

8.4 Conclusion .......................................................... 323
Bown [30] discusses the *bricoleur*-style development [129] of interactive and generative music systems in which autonomous modules are combined to form complete systems. For example, in a collaborative or workshop scenario, one participant’s machine listening software (feature extractor) might be used in conjunction with an algorithm (decision maker) developed by another participant to control a set of musical materials (generator) composed by a third participant to form a complete interactive music system. The emphasis in this type of development is on quickly experimenting with ideas by combining different modules and behaviours and it is particularly relevant in collaborative situations where the priority is to quickly create working systems that can then be used or improved.

In April 2012, the author and two colleagues (Oliver Bown¹ and Ben Carey²) organised a three-day workshop (a ‘Hacktogether’), the focus of which was to build a selection of systems for presentation in a concert that took place on the fourth day. The concert was entitled *Algorithmic Improvisations* and it took place at the Serial Space, Sydney. This was followed in June 2012 by a second concert entitled *Computer Improvisation* in which a variety of interactive and generative music systems were presented. This took place at the University of Technology, Sydney. We used the ADTK to develop two works for each of these concerts (four works in total), aiming to investigate the use of the ADTK in practise, and in particular the role it could play in the fast, bricoleur-style development of performance-ready systems. In addition to the four works presented in 2012, Bown and the author collaborated to develop a fifth system to present at a Concert in June 2013 that was part of the programme of the 9th ACM Conference on Creativity and Cognition, which also took place in Sydney.

In this chapter, we describe each of the five systems referred to above as case studies of using the ADTK. Our aim is to identify the features of the ADTK that

---

¹An electronic musician and academic researcher, also an associate supervisor of this PhD research.
²A saxophonist, electroacoustic musician and software developer.
enhance or detract from its applicability in contexts such as those described above (see research question III-(iv), Section 1.6.2). We begin with a discussion of the methodology relevant to this type of research in which the author was both an active participant in the creative work, and a researcher investigating the ADTK (Section 8.1). This is followed by details of each system including the practical and artistic context, the agent design, and the resulting performance in each case (Section 8.2). We finish with a discussion of the systems produced, the design processes undertaken and how the future development and understanding of the ADTK is informed (Section 8.3).

8.1 Methodology

As mentioned above, our aim in the research reported here was to investigate the use of the ADTK in building interactive and generative systems in real-world scenarios, and in particular, the role it could play in bricoleur-style development. As a member of the experimental music community in Sydney, it was natural for the author to participate in the creation of generative and interactive works for music performance. Of course, this had considerable bearing on the range of potential valid outcomes in our research of the ADTK, and this is discussed below.

To begin, we clarify the role of the author. For the development of four of the five systems that form the bases of the case studies presented here, the author took the role of designing the agent (i.e. designing the musical decision maker), while various collaborators developed or supplied feature extractor and generator modules and participated as acoustic instrumentalists and laptop performers. Importantly, the author took as small a role as possible in establishing musical goals for the agents, since his prior work with the ADTK had inevitably lead to certain biases about its capabilities (the types of patterns that are easy to capture, for example). One system, Isidores, is an exception to this, as it was solely the author’s work.

This research is in part an autoethnographic study (see, e.g. [77]) with the aim
of understanding the role of designers of musical decision makers, and how it is affected by using the ADTK. Relevant to this are the design process itself, the artistic context and the logistic constraints related to collaborative work. As is critical for rigorous autoethnographic research [75], we support the accounts of the various agent designs with other data: we use records of the agents themselves and the learning configurations used to create them; the training data on which the agent designs were based; and audio recordings of the agent performances. We do not make aesthetic judgements on the agents’ output, but we do comment on the agent performances insofar as they can help our understanding of the design process. In particular, we comment on the ways in which they clearly met, or not, the musical goals that informed their designs.

In addition, this research is a user-centred design study (see, e.g. [177]), aiming to better understand the needs of stakeholders in the design of interactive and generative systems, in order to arrive at new requirements and improvements for the software. Although the author was the primary user of the software, almost all of the activity that is reported took place in a collaborative context and much of it was dictated by the requirements (both creative and logistic) of other participants. By examining the ease or difficulty with which these requirements could be fulfilled, we aimed to identify a variety of improvements to the ADTK as well as avenues for further research.

Finally, we acknowledge that with regard to the autoethnographic parts of the case studies presented below, the author’s reflections on the agent design processes cannot be assumed to generalise to other would-be users of the ADTK. It is because of their limited utility that such reflections were not included as part of the study reported in the previous chapter. However, they do raise important questions for future research about the software which will lead to better designs. In addition, other aspects of the collaborative contexts reported here are not specific to the author and these point to concrete improvements that will be made to the software to better support creative work in these contexts.


8.2 Case Studies

In this section we describe separately the creation of five interactive and generative music systems. The accounts are presented in a common format. We begin by describing the context (both artistic and practical) of each work as well as the feature extractors and generators between which the musical agent was required to mediate. This is followed with a discussion of the training data and the design of the agent. Finally, each account concludes with notes on the salient features of the agent’s concert performance.

8.2.1 Automato

Context and System description

Two musical agents were developed for a concert of autonomous computer music systems entitled Algorithmic Improvisations which took place in Sydney, Australia, on April 21st, 2012. The first was designed for a piece called Automato, and the aim was that a musical agent would control an ensemble of electronic sound generators to perform in tandem with a trombonist. We describe it in this section.

The Automato system was created in collaboration with Ben Carey, and musician Astrid Zeman\(^3\). A PQfe diagram of the system (see Section 1.3.1) is shown in Figure 8.1. Both the feature extractor and generator components were supplied by Carey, having adapted them from his _derivations system which is described in detail in [43]. Only the essential details will be given here.

The feature extractor comprises a phrase detector and a phrase analyser. The phrase detector analyses the incoming audio for periods of silence longer than a pre-determined threshold. When such a period arises, it is marked as a phrase boundary (i.e. the point in time between two musical phrases played by the trombonist). Each time a phrase boundary is detected, the segment of audio corresponding to the phrase that has just ended is sent to the phrase analyser.

\(^3\)A Sydney-based multi-instrumentalist and computer scientist who took part in the workshop.
Figure 8.1: PQfe diagram of the Automato system (see Section 1.3.1). An audio signal arrives from a microphone sensing the trombonist’s performance. The feature extractor detects phrase boundaries and analyses each phrase. At the end of each phrase the analysis data is sent to the decision maker (the Automato agent) so that it can update the parameters under its control, thus modifying the output of the set of six sound generators.

The phrase analyser divides the segment into analysis frames (i.e. much shorter segments of approximately 50 ms in length) and for each frame, it calculates a number of psychoacoustically informed descriptors. Specifically, it calculates the RMS amplitude, the pitch, the brightness and the noisiness of each frame (see [43] for details). Thus, for each phrase, four vectors are calculated describing how these four quantities change over the course of the phrase (i.e. four numbers per frame). The mean and standard deviation of each of these vectors is then calculated, giving eight numbers that describe the phrase. Finally, each of these numbers is scaled and rounded so that it is represented by an integer between 1 and 10. These eight integers—corresponding to the means and standard deviations of each of the four descriptors—are transmitted to the decision maker at the end of each phrase. The decision maker uses the 8 integers describing the phrase to update the values of the parameters under its control. Note that this means that the decision points are not evenly spaced in time (they occur at phrase boundaries); the Automato system is the
Table 8.1: The music system variables of the Automato system. In the Type column, categorical is abbreviated by Cat. and Ordinal is abbreviated by Ord. The I/O column indicates if a parameter is an Input parameter which results from analysis of the trombonist’s performance; or an Output parameter which is one that the Agent controls to manipulate the sound output of the system.

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Type</th>
<th>I/O</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplitude M.</td>
<td>{1, \ldots, 10}</td>
<td>Ord.</td>
<td>Input</td>
<td>(p_1)</td>
<td>Mean Amplitude</td>
</tr>
<tr>
<td>Amplitude S.D.</td>
<td>{1, \ldots, 10}</td>
<td>Ord.</td>
<td>Input</td>
<td>(p_2)</td>
<td>Standard Deviation of Amplitude</td>
</tr>
<tr>
<td>Pitch M.</td>
<td>{1, \ldots, 10}</td>
<td>Ord.</td>
<td>Input</td>
<td>(p_3)</td>
<td>Mean Pitch</td>
</tr>
<tr>
<td>Pitch S.D.</td>
<td>{1, \ldots, 10}</td>
<td>Ord.</td>
<td>Input</td>
<td>(p_4)</td>
<td>Standard Deviation of Pitch</td>
</tr>
<tr>
<td>Brightness M.</td>
<td>{1, \ldots, 10}</td>
<td>Ord.</td>
<td>Input</td>
<td>(p_5)</td>
<td>Mean Brightness</td>
</tr>
<tr>
<td>Brightness S.D.</td>
<td>{1, \ldots, 10}</td>
<td>Ord.</td>
<td>Input</td>
<td>(p_6)</td>
<td>Standard Deviation of Brightness</td>
</tr>
<tr>
<td>Noisiness M.</td>
<td>{1, \ldots, 10}</td>
<td>Ord.</td>
<td>Input</td>
<td>(p_7)</td>
<td>Mean Noisiness</td>
</tr>
<tr>
<td>Noisiness S.D.</td>
<td>{1, \ldots, 10}</td>
<td>Ord.</td>
<td>Input</td>
<td>(p_8)</td>
<td>Standard Deviation of Noisiness</td>
</tr>
<tr>
<td>Pvoc Bongo</td>
<td>{0, 1}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_9)</td>
<td>Toggle sound generator on/off</td>
</tr>
<tr>
<td>Pvoc Tchk</td>
<td>{0, 1}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_{10})</td>
<td>Toggle sound generator on/off</td>
</tr>
<tr>
<td>Pvoc Guitar</td>
<td>{0, 1}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_{11})</td>
<td>Toggle sound generator on/off</td>
</tr>
<tr>
<td>Pvoc Salad</td>
<td>{0, 1}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_{12})</td>
<td>Toggle sound generator on/off</td>
</tr>
<tr>
<td>Pitch Model</td>
<td>{0, 1}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_{13})</td>
<td>Toggle sound generator on/off</td>
</tr>
<tr>
<td>Granulator</td>
<td>{0, 1}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_{14})</td>
<td>Toggle sound generator on/off</td>
</tr>
<tr>
<td>Pitch M Preset</td>
<td>{1, 3, 4, 5, 6, 7}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_{15})</td>
<td>Mode of Pitch Model</td>
</tr>
</tbody>
</table>

only system described in this thesis with this characteristic.

The decision maker controls six binary-valued variables and an integer-valued variable. The six binary-valued variables represent the states of the on/off switches for the six sound generators (again, see Figure 8.1). Each of the sound generators manipulates pre-recorded audio and is controlled at a lower level by various random processes (again, details are given in [43]). The integer-valued variable parameterises one of the sound generators (the Pitch Model). A summary of the music system variables (both those input to the decision maker from the feature extractor, and those output to the generator from the decision maker) is given in Table 8.1.
Training Data and Agent Design

To design an agent, a set of five training examples were recorded in which Carey controlled the ensemble of sound generators alongside Zeman, playing trombone. Each time a trombone phrase ended, a snapshot was recorded of the phrase descriptors and the values of the variables under Carey’s control. A series of such snapshots comprises an example performance. The data from the example performances are shown in Figure 8.2. In total there are 232 decision points in the data set.

As mentioned above, the Automato system was created as part of a three-day workshop on developing autonomous music systems. Due to the number of projects being pursued, the busy schedules of the participants and the high demand on rehearsal space, there were severe time constraints on the agent design process. The five example performances were each between three and four minutes in length, approximately, and they were recorded in just over a half hour. The agent was then designed in approximately one hour, in which it was possible to audition the agent’s behaviour playing alongside pre-recorded trombone, but not interactively with the performer.

To begin the design process, four basic stylistic requirements for the agent were established through discussions with Carey and Zeman. They were as follows:

1. There should always be at least one sound generator active.
2. There should be dynamic variation.
3. There should be reasonable variety without seeming ‘random’ (i.e. without changing incoherently from one set of variable values to the next).
4. There should be some interactivity; the agent should respond to certain phrases or types of phrases played on the trombone.

Interestingly, the first of these was only added after it was suggested by the author; it had not occurred to the other participants that the agent might do such an obviously inappropriate thing as to stop playing entirely (i.e. choose the value 0 for parameters
Figure 8.2: The training data for the Automato system. There are five training examples, (a)-(e). In each one, the bottommost eight rows correspond to the trombone analysis variables ($p_1$, ..., $p_8$). Above these are the six binary-valued variables indicating which of the six sound generators were active ($p_9$, ..., $p_{14}$) and an integer-valued variable corresponding to a parameter of one of the sound generators ($p_{15}$). Each column corresponds to a decision point.
Also noteworthy is that the other three requirements are quite high-level and translating them to concrete design steps is not necessarily straightforward. Both of these issues will be revisited in Section 8.3. Finally, we did not identify any particular stylistically inappropriate parameter combinations to be avoided by the agent.

The learning configuration at which we arrived is shown in Table 8.2 and we give details in the following. In order to ensure that there would always be at least one sound generator active, we created a custom variable, \( p_{16} \), equal to the sum of \( \{ p_9, \ldots, p_{14}, \} \), i.e. it counts the number of sound generators that are switched on. Simply including \( p_{16} \) in the system places appropriate bounds on the number of active sound generators, since its domain is \( \{1, \ldots, 6\} \) (i.e. it cannot have a value of 0). However, with the aim of introducing dynamic variation (item 2 above), we applied a 10th-order VMM to \( p_{16} \) so that the number of active sound generators would vary in a way that is similar to the examples.

To introduce variety without seeming random (item 3 above), we also applied 5th-order VMMs to each of the music system output variables (those controlled by the agent to directly parameterise the sound generators). The \( \text{SUM} \) variable, \( p_{16} \) was given a higher priority than the individual music system variables, so that it would take precedence.

As mentioned above, the example performances were created very quickly and except for a short period of testing prior to creating them, these were the first times that Carey and Zeman had played together. This is typical of the spontaneous collaborations that arise in events such as the Hacktogether; improvisation is frequently entered into blind, with no preparation or rehearsal. Indeed, this one of the many reasons why such events are of creative interest. However, one of the consequences of this was that the duo had not developed a particular characteristic playing style that might include specific responses or actions on the part of either player. Thus, there were no stylistic requirements relating to the interaction between the two players. In addition to this, our method for representing trombone phrases had
Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Active Generators</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( p_{16} = \sum (p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}) )</td>
<td></td>
</tr>
<tr>
<td>Pitch Model</td>
<td>( p_{13} )</td>
<td>5</td>
</tr>
<tr>
<td>Granulator</td>
<td>( p_{14} )</td>
<td>5</td>
</tr>
<tr>
<td>P voc Bongo</td>
<td>( p_9 )</td>
<td>5</td>
</tr>
<tr>
<td>P voc Guitar</td>
<td>( p_{11} )</td>
<td>5</td>
</tr>
<tr>
<td>P voc T chk</td>
<td>( p_{10} )</td>
<td>5</td>
</tr>
<tr>
<td>Pitch M Preset</td>
<td>( p_{15} )</td>
<td>5</td>
</tr>
</tbody>
</table>

Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>( { S_{\text{min}}, C_{\text{min}}, #_{\text{max}} } )</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactivity</td>
<td>{0.05, 0.99, 3}</td>
<td>{( p_1, \ldots, p_8, p_9, p_{11}, \ldots, p_{15} )}</td>
</tr>
</tbody>
</table>

Table 8.2: The learning configuration for the Automato musical agent. The variables are listed in priority order (see Section 5.4.5) with the highest-priority variable at the top. For custom variables, the labels, formulae (see Section 5.4.6, particularly Table 5.8 for an explanation of the notation) and VMM orders are shown, while for other (‘non-custom’) variables the formulae are replaced by the appropriate symbols. For each rule group, the the minimum support, \( S_{\text{min}} \), the minimum confidence, \( C_{\text{min}} \), and the maximum itemset size, \( #_{\text{max}} \), (see Section 5.4.4) are shown; along with a list of the variables included in the group.
not been proven to correspond veridically to features relevant to Carey’s musical
decision making. However, in the spirit of bricoleur-style system development,
and in particular, its application to the development of dynamical systems without
prescribed musical elements (as in, for example, the systems described in [30]), we
considered the trombone analysis variables merely as a means to provide informa-
tion about the acoustic environment that could influence a musical agent’s decision
making. In other words, our aim was to introduce some responsiveness into the
agent’s performance, and not specific responses.

With this in mind, we experimented with different rule groups in order to ar-
rive at a learning configuration that would give rise to rules which in turn would
introduce responsiveness to the agent’s behaviour. In order to quickly gauge the
efficacy of a rule group (i.e. without training the agent and auditioning it), we
adopted a strategy whereby we attempted to infer the effects of a given set of rules
by inspecting them visually (the rules are printed to the screen in the software). Our
goal was to find rule groups that would give rise to a modestly sized set of rules (not
so large as to be too restrictive), that would allow the trombone analysis variables to
affect the agent’s decision making.

For small sets of rules, visual inspection provided a convenient way to quickly
gain an understanding of how the agent would behave. However, one feature of
the Apriori rule-learning algorithm used in the ADTK is that it learns all rules,
including redundant ones (i.e. rules that can be accounted for by simpler rules). One
consequence of this is that if a particular variable, say $x$, retains the same value, $c$,
most of the time, it is likely that many rules will be found in which the consequent is
$x = c$, despite the fact that most of these rules contain no useful information. When
this occurs, the list of rules printed in the software can be very long and it is difficult
to gain an understanding of the effects of the rules by inspecting it, because it is
difficult to visually filter out the redundant rules. This arose during the design of the
Automato agent. Since the P voc Tchk generator (one of the software sound sources)
was used very little in the example performances, this variable was 0 most of the
time (see Figure 8.2). Thus, when included in a rule group, many rules arose with the consequent $\text{Pvoc Tchk} = 0$. For this reason the $\text{Pvoc Tchk}$ variable was omitted from the rule group that we selected, since its inclusion gave rise to a large set of rules, the likely effects of which could not be easily understood by inspection. This is noteworthy because, in retrospect, the learning configuration was chosen in part because the resulting model was easier to understand (it had fewer rules), rather than because it was necessarily better.

Both the selection of variables to include in the rule group and the rule-learning parameters (see Section 5.4.4) were chosen by iteratively re-configuring a rule group and inspecting the resulting rules. After a number of iterations, we arrived at a rule group (labelled ‘Interactivity’ in Table 8.2) that gave rise to the discovery of 31 rules, 17 of which had phrase descriptor variables in the antecedent (i.e. 17 of which could potentially result in a decision made by the agent being affected by the trombonist’s playing behaviour). Three examples of the rules discovered were

$$\text{Amplitude M} = 6 \ \text{AND Amplitude S.D.} = 2 \implies \text{Granulator} = 0,$$

$$\text{Brightness S.D.} = 0 \ \text{AND Pitch M Preset} = 6 \implies \text{Pvoc Bongo} = 1,$$

$$\text{Amplitude M} = 6 \ \text{AND Pitch Model} = 0 \implies \text{Pvoc Salad} = 0.$$

The agent trained using this rule group was auditioned alongside pre-recorded trombone. It seemed to adhere to the stylistic requirements (items 1-4 in the list above) and to be responsive to the trombone playing.

**Agent Performance**

The live performance in which the trombonist improvised alongside the agent lasted approximately three and a half minutes and was documented with an audio recording\(^4\). With reference to the list of stylistic requirements given in the previous section, we note that there was at least one sound generator active throughout (item

\(^4\)Available at: soundcloud.com/runciblenoise/trombone-and-live-electronics.
1) and there was clear dynamic variation (item 2) in the agent’s performance. In addition, the agent’s performance included a variety of different combinations of generators while arguably maintaining short-term continuity (item 3; it did not seem to change incoherently from one parameter configuration to the next). The agent did appear to respond to the trombonist’s performance (item 4) but from the audio alone it is difficult to identify the effects of particular rules; certainly, some of the apparent responsiveness is due to the fact that the agent parameter updates are triggered at the boundaries between phrases played by the trombonist.

One interesting feature of the agent’s performance was its frequent use of the \textit{Pvoc Tchk} generator. As mentioned above, this was used rarely in the example performances, and this was because its sound output was somewhat overpowering and contrasted aesthetically with the output of the other generators. Likewise, it was used only rarely when we auditioned the agent with pre-recorded trombone. However, when the real trombone was introduced (during the concert) this generator was used quite frequently. In retrospect it might have been better to omit it, or modify its characteristics to produce different output. However, here it serves to highlight a number of aspects of the design process. First, there is a need in some circumstances to articulate ‘negative’ stylistic requirements, that is, requirements that certain events do not occur, and furthermore, these may not easily come to mind. Second, the agent’s performance shows a discrepancy between our understanding of the agent’s model of musical performance and the model itself. We return to this issue in Section 8.3. Finally, we note that it is possible that had we not excluded the \textit{Pvoc Tchk} variable from the ‘Interactivity’ rule group (as we did for the reasons outlined above), rules might have been discovered to prevent the overuse of this generator.
Rhodes
Drums
Other Systems
Many Piano Chops

Figure 8.3: Sketch of the planned structure of the improvisation involving the Many Piano Chops system. Rhodes refers to the Fender Rhodes player, Drums refers to the drummer, Other Systems refers to other autonomous systems and Many Piano Chops is the system being described here. The coloured bars indicate when these players and systems were to be active during the performance.

8.2.2 Many Piano Chops

Context and System Description

The Many Piano Chops agent was the second musical agent developed using the ADTK for the Algorithmic Improvisations concert. It was intended to take part in an extended improvisation involving two live instrumentalists and two other autonomous music systems. Specifically, it was intended to play during the second half of the performance with the other autonomous systems having played during the first half, approximately, thereby providing long time scale structure to the performance (see Figure 8.3). In addition, rather than taking the role of an autonomous improviser, the Many Piano Chops system was designed to create a gradually evolving sonic backdrop against which the instrumentalists could improvise.

The Many Piano Chops agent was created in collaboration with Oliver Bown. It was a generative agent, having no feature extractor (i.e. it had no inputs), and it was designed to control an Ableton Live set comprising six tracks of musical material (i.e. up to six sounds could be played simultaneously) and a variety of audio effects controlled using seven binary-valued parameters (toggle switches). An important characteristic of the Ableton Live set is that three of the tracks contain a large amount of musical material, having close to 60 clips in each one (this gave rise to the title of the system). A summary of the music system variables to be controlled by the Many
Table 8.3: The music system variables of the Many Piano Chops system. In the Type column, categorical is abbreviated by Cat. and Ordinal is abbreviated by Ord. The domains of variables corresponding to Ableton Live tracks include the clip numbers (from 0 upwards) and the value -2 which indicates that no clip is playing in the track.

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Type</th>
<th>I/O</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piano 1 Track</td>
<td>$[−2, 0 \ldots 56]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_1$</td>
<td>Clip playing</td>
</tr>
<tr>
<td>Piano FX Track</td>
<td>$[−2, 0 \ldots 58]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_2$</td>
<td>Clip playing</td>
</tr>
<tr>
<td>Piano EQ Track</td>
<td>$[−2, 0 \ldots 57]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_3$</td>
<td>Clip playing</td>
</tr>
<tr>
<td>Scratch Track</td>
<td>$[−2, 0 \ldots 8]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_4$</td>
<td>Clip playing</td>
</tr>
<tr>
<td>Scratch FX Track</td>
<td>$[−2, 0 \ldots 14]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_5$</td>
<td>Clip playing</td>
</tr>
<tr>
<td>Kick Drum Track</td>
<td>$[−2, 0 \ldots 3]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_6$</td>
<td>Clip playing</td>
</tr>
<tr>
<td>PFX Reverb</td>
<td>$[0, 1]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_7$</td>
<td>Toggle effect on/off</td>
</tr>
<tr>
<td>PFX Rev.</td>
<td>$[0, 1]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_8$</td>
<td>Toggle effect on/off</td>
</tr>
<tr>
<td>PFX Freq Shift</td>
<td>$[0, 1]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_9$</td>
<td>Toggle effect on/off</td>
</tr>
<tr>
<td>PFX Vocoder</td>
<td>$[0, 1]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{10}$</td>
<td>Toggle effect on/off</td>
</tr>
<tr>
<td>SFX Low</td>
<td>$[0, 1]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{11}$</td>
<td>Toggle parameter on/off</td>
</tr>
<tr>
<td>SFX Mid</td>
<td>$[0, 1]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{12}$</td>
<td>Toggle parameter on/off</td>
</tr>
<tr>
<td>SFX High</td>
<td>$[0, 1]$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{13}$</td>
<td>Toggle parameter on/off</td>
</tr>
</tbody>
</table>

Piano Chops agent is given in Table 8.3.

**Training Data and Agent Design**

As in the case of the Automato agent, there were severe constraints on the amount of time to record examples and then design Many Piano Chops agent. The agent was trained using a single example comprising 117 decision points and lasting approximately 6 minutes and 15 seconds (see Figure 8.4). These data are plotted differently from other plots of Ableton Live performance data in this thesis, because of the large number of clips on three of the tracks, and the musical importance of the manner in which they were sequenced.

We compiled the following list of stylistic requirements for the agent’s performance:

1. For the three Piano tracks ($p_1, p_2, p_3$), the agent should gradually progress from the lowest numbered clips to the highest numbered clips. Periods of silence are appropriate, though there should always be at least one of these three tracks
Figure 8.4: The training data for the Many Piano Chops system. The three plots together show the single training example. The first, Piano Tracks, shows the clips that were used over the course of the performance in the first three Ableton Live tracks. The second, Other Tracks, shows the clips that were used over the course of the performance in the remaining Ableton Live tracks. Finally, the third plot, Effects Parameters, shows the values of seven binary-valued variables over the course of the performance.
2. The *Scratch, Scratch FX* and *Kick Drum* tracks should be used, particularly as the performance progresses, but the clips should not change too quickly (i.e. the general rate of change shown in the example should be adhered to).

3. The audio effect parameters \( p_7, \ldots, p_{13} \) should be used intermittently for variety but they are not important with respect to the overall musical structure.

4. The audio effect parameters *SFX Low, SFX Med* and *SFX High* \( (p_{11}, p_{12}, p_{13}) \) control a filter effect. If they are all ‘off’ (i.e. if they all have value 0) then no sound is heard from the *Scratch FX* track. Thus, at least one of these three parameters must have a value of 1 at all times.

For the design of the *Many Piano Chops* agent, we followed that of the *Automato* agent by adopting strategies to mitigate the need to repeatedly audition agent performances. The example performance on which the agent was to be based was over 6 minutes in length, so repeatedly listening to agent performances would have been fatiguing. Instead, we considered each stylistic requirement in turn and attempted to determine ways of augmenting the learning configuration in order to incorporate it into the agent’s behaviour. The strategy of inspecting learnt rules, used previously, was also employed.

The learning configuration arrived at for the *Many Piano Chops* agent is given in Table 8.4. The first three custom variables \( (p_{14}, p_{15} \text{ and } p_{16}) \) represent the clip being played in each of the three piano tracks as a number between 0 and 4, where 0 means that there is no clip playing; 1 means that the clip number is between 0 and 15; 2 means that it is between 16 and 30; 3 means that it is between 31 and 45; and finally 4 means that the clip number is greater than 45. That is, these custom variables coarsely quantize the number of the clip being played into five regions (0-5) in order to capture the longer time scale pattern by which the agent should gradually move towards higher-numbered clips (item 1 above), without simply copying the training example. These three custom variables, together with *Any Scratch On* \( (p_{17}) \) and *Kick*
Drum On \((p_{18})\), are included in the Structural Variables rule group in order to learn rules describing broad relationships between the six Ableton Live tracks (items 1 and 2 above). High-order VMMs are also used to model these variables to capture the temporal structure in the example performance (items 1 and 2). Lower-order VMMs are used to model the individual music system variables, which have lower priorities than the aforementioned custom variables (items 1, 2 and 3 above). Finally, the seven variables associated with the audio effects \((p_7, \ldots, p_{13})\) are included in the Effects Parameters rule group so that rules ensuring that one of the filter effect variables is always on (item 4 above).

**Agent Performance**

The Many Piano Chops agent was used during an improvisation between a percussionist using a drum kit and a keyboardist playing a Fender Rhodes electric piano. The performance was documented with an audio recording\(^5\). The Many Piano Chops agent was started after around four and a half minutes and it could be heard to gradually progress through the large number of clips in the first three Ableton Live tracks. The output was perhaps more sparse than intended; the Scratch Track and Scratch FX Track were not frequently used, and the Kick Drum Track was not used at all. These elements may have been used more if the performance had developed further, but the trio comprising the agent and the two musicians arrived at a natural ending after approximately eight and a half minutes in total. This highlights the creative interest of using such a generative system in an improvisational context: as well as providing a sonic backdrop, it can demand responses from the musicians and potentially alter the course of a performance.

\(^5\)Available at: soundcloud.com/runciblenoise/rhodes-drums-and-two.
### Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piano 1 Track Clip Region</td>
<td>$p_{14} = \left[ \Sigma(&gt;<em>{\text{ANY}} (-1, p_1), &gt;</em>{\text{ANY}} (15, p_1),&gt;<em>{\text{ANY}} (30, p_1), &gt;</em>{\text{ANY}} (45, p_1)) \right]$</td>
</tr>
<tr>
<td>Piano FX Track Clip Region</td>
<td>$p_{15} = \left[ \Sigma(&gt;<em>{\text{ANY}} (-1, p_2), &gt;</em>{\text{ANY}} (15, p_2),&gt;<em>{\text{ANY}} (30, p_2), &gt;</em>{\text{ANY}} (45, p_2)) \right]$</td>
</tr>
<tr>
<td>Piano EQ Track Clip Region</td>
<td>$p_{16} = \left[ \Sigma(&gt;<em>{\text{ANY}} (-1, p_3), &gt;</em>{\text{ANY}} (15, p_3),&gt;<em>{\text{ANY}} (30, p_3), &gt;</em>{\text{ANY}} (45, p_3)) \right]$</td>
</tr>
<tr>
<td>Any Scratch On</td>
<td>$p_{17} = \left[ &gt;_{\text{ANY}} (-1, p_4, p_5) \right]$</td>
</tr>
<tr>
<td>Kick Drum On</td>
<td>$p_{18} = \left[ &gt;_{\text{ANY}} (-1, p_6) \right]$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piano 1 Track</td>
<td>8</td>
</tr>
<tr>
<td>Piano EQ Track</td>
<td>8</td>
</tr>
<tr>
<td>Piano FX Track</td>
<td>8</td>
</tr>
<tr>
<td>Scratch Track</td>
<td>10</td>
</tr>
<tr>
<td>Kick Drum Track</td>
<td>8</td>
</tr>
<tr>
<td>PFX Reverb</td>
<td>1</td>
</tr>
<tr>
<td>PFX Rev. Freeze</td>
<td>2</td>
</tr>
<tr>
<td>PFX Freq Shift</td>
<td>2</td>
</tr>
<tr>
<td>PFX Vocoder</td>
<td>2</td>
</tr>
<tr>
<td>SFX Low</td>
<td>2</td>
</tr>
<tr>
<td>SFX Med</td>
<td>2</td>
</tr>
<tr>
<td>SFX High</td>
<td>2</td>
</tr>
</tbody>
</table>

### Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>${S_{\text{min}}, C_{\text{min}}, #_{\text{max}}}$</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Variables</td>
<td>${0.01,0.99,3}$</td>
<td>${p_{14}, \ldots, p_{18}}$</td>
</tr>
<tr>
<td>Effects Parameters</td>
<td>${0.01,0.99,3}$</td>
<td>${p_{7}, \ldots, p_{13}}$</td>
</tr>
</tbody>
</table>

Table 8.4: The learning configuration for the Many Piano Chops musical agent. The variables are listed in priority order (see Section 5.4.5) with the highest-priority variable at the top. For custom variables, the labels, formulae (see Section 5.4.6, particularly Table 5.8 for an explanation of the notation) and VMM orders are shown, while for other ('non-custom') variables the formulae are replaced by the appropriate symbols. For each rule group, the the minimum support, $S_{\text{min}}$, the minimum confidence, $C_{\text{min}}$, and the maximum itemset size, $#_{\text{max}}$, (see Section 5.4.4) are shown; along with a list of the variables included in the group.
8.2. Case Studies

8.2.3 Isidores

Context and System Description

The Isidores system is the first of two that were presented at the Computer Improvisation concert that took place in June, 2012. It is a generative composition, and unlike the other works reported in this chapter, it is entirely the work of the author. In the following, we describe the system, with reference to Figure 8.5 and Table 8.5, which give the details of the music system variables.

The Isidores system comprises a Max patch in which sound processors and synthesizers are controlled by (i) low-level probabilistic processes that generate rhythmic patterns of musical control data, and (ii) sequencers that playback pre-composed sequences of musical control data. For succinctness, we refer to the former as probabilistic sequencers (abbreviated by Prob Seq in Figure 8.5) and the latter as deterministic sequencers (abbreviated by Det Seq in Figure 8.5). There are four probabilistic sequencers which are used to control a set of four player modules, each of which produces sound by manipulating a pre-loaded sound file.

The four probabilistic sequencers, respectively, are controlled by music system
<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Type</th>
<th>I/O</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq 1 Mode</td>
<td>{0, 1, 2}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_1)</td>
<td>Sequencer 1 Mode</td>
</tr>
<tr>
<td>Seq 2 Mode</td>
<td>{0, 1, 2}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_2)</td>
<td>Sequencer 2 Mode</td>
</tr>
<tr>
<td>Seq 3 Mode</td>
<td>{0, 1, 2}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_3)</td>
<td>Sequencer 3 Mode</td>
</tr>
<tr>
<td>Seq 4 Mode</td>
<td>{0, 1, 2}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_4)</td>
<td>Sequencer 4 Mode</td>
</tr>
<tr>
<td>Player 2 Mode</td>
<td>{1, 2}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_5)</td>
<td>Player 2 Mode</td>
</tr>
<tr>
<td>Player 3 Mode</td>
<td>{1, 2}</td>
<td>Cat.</td>
<td>Output</td>
<td>(p_6)</td>
<td>Player 3 Mode</td>
</tr>
<tr>
<td>Player 4 Mode</td>
<td>{1, 2}</td>
<td>Ord.</td>
<td>Output</td>
<td>(p_7)</td>
<td>Player 4 Mode</td>
</tr>
<tr>
<td>Melody Synth Mode</td>
<td>{0, 1, 2}</td>
<td>Ord.</td>
<td>Output</td>
<td>(p_8)</td>
<td>Melody Synth Mode</td>
</tr>
<tr>
<td>Bass Toggle</td>
<td>{0, 1}</td>
<td>Ord.</td>
<td>Output</td>
<td>(p_9)</td>
<td>Toggle Bass On/Off</td>
</tr>
<tr>
<td>Ambient Synth Mode</td>
<td>{0, 1, 2}</td>
<td>Ord.</td>
<td>Output</td>
<td>(p_{10})</td>
<td>Ambient Synth Mode</td>
</tr>
<tr>
<td>Effects Routing Config</td>
<td>{1, 2, 3}</td>
<td>Ord.</td>
<td>Output</td>
<td>(p_{11})</td>
<td>Routing from Players to Effects</td>
</tr>
<tr>
<td>Control Routing Config</td>
<td>{1, 2, 3}</td>
<td>Ord.</td>
<td>Output</td>
<td>(p_{12})</td>
<td>Routing from Sequencers to Players</td>
</tr>
</tbody>
</table>

Table 8.5: The music system variables of the Isidores system. In the Type column, categorical is abbreviated by Cat. and Ordinal is abbreviated by Ord.

variables \(p_1, \ldots, p_4\) which may take the value 0, meaning that the sequencer is inactive; 1, meaning that it is operating according to one set of time- and probability-related parameters; and 2, meaning that it is operating according to another set of such parameters. Of the four player modules, the first has no music system parameters associated with it, while each of the other three has two output modes that determine the characteristics of their output (the music system variables \(p_5, p_6, p_7\) control the modes of Player modules 2, 3 and 4, respectively). Finally, the control values from the four sequencers may be routed to the four players in three different ways as determined by the Control Routing Config variable, \(p_{12}\). Specifically, each sequencer may be connected to a single Player (sequencer 1 to Player 1, sequencer 2 to Player 2, etc.); or all Player modules can be controlled simultaneously by sequencer 4; or finally, the sequencers are connected to the Player modules in reverse order (sequencer 1 to Player 4, sequencer 2 to Player 3, etc.). This is illustrated in Figure 8.5.

The audio signals from the four players can be heard directly or passed through various audio effects. The Effects Routing Config controls the routing between the
Player modules and the audio effects. There are three possible routings ($p_{11}$).

Finally, there are three synthesizers, each of which is controlled by a single music system variable. The Melody Synth Mode ($p_{8}$) controls the sequence of notes being output by the Melody Synth. Next, the Bass Toggle ($p_{9}$) controls whether or not the Bass Synth is active. Finally, the Ambient Synth Mode ($p_{10}$) controls the sequence of notes being output by the Ambient Synth (see Figure 8.5).

**Training Data and Agent Design**

We began the agent design process by creating a set of example performances (five in all, totalling 359 decision points; see Figure 8.6). Upon consideration of the Isidores system, and reflecting on the example performances, we elected to create an ‘experimental’ agent that would not be too constrained by stylistically salient patterns in the example performances. Almost all of the musical material worked together (i.e. almost all combinations of music system variable values were appropriate) so we designed an agent that would broadly exhibit dynamic variation while being minimally constrained in other ways.

The learning configuration used is shown in Table 8.4 and we give details here. Following a technique used for the Automato agent, a SUM custom variable, Number of Active Sources ($p_{13}$), was used to represent the number of active sound sources. Its underlying variables included only the four sequencer modules, the melody synthesizer and the ambient synthesizer. The bass synthesizer was not included because its output was too low in pitch and level to be contribute significantly to the overall dynamics of the music. The Number of Active Sources custom variable was given the highest priority and modelled using a 5th-order VMM. All other variables are given lower priorities and are modelled with 1st-order VMMs.

The second custom variable, Drum Echo ($p_{14}$), was created along with the Prevent Drum Echo rule group in order to ensure that one particular configuration of the music system variables could not arise. The configuration in question includes particular values of the of the Effects Routing Config variable, the Player 2 Mode
Figure 8.6: The training data for the Isidores system. There are five examples in total, totalling 359 decision points.
### Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Active Sources</td>
<td>( p_{13} = \left[ \Sigma (＞<em>{ANY} (0, p_1), ＞</em>{ANY} (0, p_2), ＞<em>{ANY} (0, p_3), ＞</em>{ANY} (0, p_4), ＞<em>{ANY} (0, p_8), ＞</em>{ANY} (0, p_{10}) ) \right] )</td>
<td>5</td>
</tr>
<tr>
<td>Seq 2 Mode</td>
<td>( p_2 )</td>
<td>1</td>
</tr>
<tr>
<td>Player 2 Mode</td>
<td>( p_5 )</td>
<td>1</td>
</tr>
<tr>
<td>Melody Synth Mode</td>
<td>( p_8 )</td>
<td>1</td>
</tr>
<tr>
<td>Seq 3 Mode</td>
<td>( p_3 )</td>
<td>1</td>
</tr>
<tr>
<td>Seq 4 Mode</td>
<td>( p_4 )</td>
<td>1</td>
</tr>
<tr>
<td>Seq 1 Mode</td>
<td>( p_1 )</td>
<td>1</td>
</tr>
<tr>
<td>Ambient Synth Mode</td>
<td>( p_{10} )</td>
<td>1</td>
</tr>
<tr>
<td>Effects Routing Config</td>
<td>( p_{11} )</td>
<td>1</td>
</tr>
<tr>
<td>Control Routing Config</td>
<td>( p_{12} )</td>
<td>1</td>
</tr>
<tr>
<td>Bass Toggle</td>
<td>( p_9 )</td>
<td>1</td>
</tr>
<tr>
<td>Player 3 Mode</td>
<td>( p_6 )</td>
<td>1</td>
</tr>
<tr>
<td>Player 4 Mode</td>
<td>( p_7 )</td>
<td>1</td>
</tr>
<tr>
<td>Drum Echo</td>
<td>( p_{14} = \left[ == (3, p_{11}) \right] )</td>
<td>0</td>
</tr>
</tbody>
</table>

### Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>( { S_{min}, C_{min}, #_{max} } )</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound Producers</td>
<td>{0.1, 0.9, 4}</td>
<td>{( p_5, \ldots, p_{10} )}</td>
</tr>
<tr>
<td>Control and Routing</td>
<td>{0.1, 0.9, 4}</td>
<td>{( p_1, \ldots, p_4, p_{12} )}</td>
</tr>
<tr>
<td>Prevent Drum Echo</td>
<td>{0.1, 0.999, 4}</td>
<td>{( p_2, p_5, p_{14} )}</td>
</tr>
</tbody>
</table>

Table 8.6: The learning configuration for the Isidores musical agent. The variables are listed in priority order (see Section 5.4.5) with the highest-priority variable at the top. For custom variables, the labels, formulae (see Section 5.4.6, particularly Table 5.8 for an explanation of the notation) and VMM orders are shown, while for other (‘non-custom’) variables the formulae are replaced by the appropriate symbols. For each rule group, the the minimum support, \( S_{min} \), the minimum confidence, \( C_{min} \), and the maximum itemset size, \( #_{max} \) (see Section 5.4.4) are shown; along with a list of the variables included in the group.
variable and the \textit{Seq 2 Mode} variable that lead to a rapid drum beat being passed through a echo effect and giving rise to an unpleasant booming sound. Thus, to prevent this from happening (it did not happen in the training data) $p_{14}$ was created to indicate if the \textit{Effects Routing Config} was set to the value 3 (the problematic value) and it was added to a rule group (‘Prevent Drum Echo’; see Table 8.6), resulting in rules that amounted to:

\[
\text{Seq 2 Mode} = 2 \text{ AND } \text{Player 2 Mode} = 1 \implies \text{Effects Routing Config} \neq 3.
\]

This rule prevents the problematic variable values from arising.

Finally, two additional rule groups (‘Sound Producers’ and ‘Control and Routing’; again see Table 8.4) were added. This was done with the idea of introducing very limited constraints to the combinations of variables available to the agent. As in previous cases, these particular rule groups were arrived at iteratively using both visual inspection of the learnt rules and agent auditioning to guide the process.

\section*{Agent Performance}

A performance of the Isidores system was presented at the aforementioned \textit{Computer Improvisers} concert. It was documented with an audio recording\footnote{Available at: soundcloud.com/runciblenoise/isidores-performance.}. A small problem arose soon after the performance began when all instruments were turned off for a period of approximately eight seconds. While logs of the agent’s decisions were not recorded, subsequent analysis showed that it is possible for the agent to set the \textit{Control Routing Config} parameter such that all player modules are being controlled by the fourth sequencer module and to simultaneously set the fourth sequencer module to ‘off’. This means that unless one of the three synthesizers is playing, no instruments will sound. It would have been straightforward to prevent this using an appropriately configured rule group, and that this was not done was an oversight in the agent’s design.
8.2.4 Cello Improvisation System

Context and System Description

The second agent created for the Computer Improvisers concert was designed to control an Ableton Live set alongside an improvising cellist. The improvisation was untitled and in the following, we refer to the system comprising the agent and Ableton Live set as the Cello Improvisation system. The Ableton Live set was created by Oliver Bown, who also designed the agent in collaboration with the author.

The Ableton Live set had 20 tracks to be controlled by the agent. Each track contained clips containing audio segments, some of which were looped (i.e. once triggered, the clip would play repeatedly until the track was stopped or another clip was triggered) and some of which were not (i.e. when triggered, the clip would play once only). The music system variables are shown in Table 8.7 in which the first 20 rows ($p_1, \ldots, p_{20}$) correspond to the Ableton Live tracks. The domain of each variable indicates which clips were available in each track (as previously, the value -2 corresponds to the state of a track state when no clip is playing).

To enable the agent to respond to the cellist, a modified version of the Agent Designer Device (the component that allows the ADTK to be used inside Ableton Live; see Section 5.5) was created with some rudimentary machine listening capabilities. Jehan’s analyzert object (i.e. plugin) for Max was used to calculate psychoacoustically-informed descriptors of the incoming cello signal. Four descriptors were calculated for each 2048-sample frame of the signal: pitch, loudness, brightness and noisiness and additionally the frame was classified as containing an onset (i.e. the beginning of a new note) or not. Thus, for each musical bar, four vectors were calculated describing how each of the four descriptor quantities changed over the course of the bar, and in addition, the total number of onsets that occurred in the bar was counted. This allowed five values to be calculated at each decision point: the mean of each of the four descriptor vectors over the previous bar and the total number of onsets over

7See: web.media.mit.edu/~tristan/maxmsp.html
<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Type</th>
<th>I/O</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organ 1</td>
<td>${-2, 0, 1, 5, \ldots, 8, 10}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_1$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Organ 2</td>
<td>${-2, 1, 2, 6, \ldots, 8}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_2$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Organ 3</td>
<td>${-2, 1, 8, 10, 12}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_3$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Organ 4</td>
<td>${-2, 0, 2, 8, 10}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_4$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Piano 1</td>
<td>${-2, 0, 1, 3, 5, 7, 12}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_5$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Piano 2</td>
<td>${-2, 0, 5, 10}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_6$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Bass</td>
<td>${-2, 0, 1, 5, 7, 9}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_7$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Rust 1</td>
<td>${-2, 1, 2, 4, 8}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_8$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Rust 2</td>
<td>${-2, 0, \ldots, 3, 6, \ldots, 10, 12, 14}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_9$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Rust Patter</td>
<td>${-2, 1, 2, 4, 6, 8, 10, 12}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{10}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Rust 4</td>
<td>${-2, 2, 7}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{11}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Samples 1</td>
<td>${-2, 2, 3, 4, 6, 7, 8}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{12}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Samples 2</td>
<td>${-2, 0, 2, 5, 6, 8, 10, 11}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{13}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Tuba</td>
<td>${-2, 0, 2, 3}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{14}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Strat</td>
<td>${-2, 1, 3, 5, 8}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{15}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Samples 3</td>
<td>${-2, 1, 2, 4, 6, 9, \ldots, 12}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{16}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Samples 4</td>
<td>${-2, 1, 2, 4, 5, 8, 9, 12}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{17}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Samples 5</td>
<td>${-2, 1}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{18}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Samples 6</td>
<td>${-2, 6, 9, 11}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{19}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Samples 7</td>
<td>${-2, 0, 3, 5, 7, 8, 10, 12}$</td>
<td>Cat.</td>
<td>Output</td>
<td>$p_{20}$</td>
<td>Clip selection</td>
</tr>
<tr>
<td>Pitch</td>
<td>${0, \ldots, 5}$</td>
<td>Ord.</td>
<td>Input</td>
<td>$p_{21}$</td>
<td>Cello analysis</td>
</tr>
<tr>
<td>Loudness</td>
<td>${0, \ldots, 5}$</td>
<td>Ord.</td>
<td>Input</td>
<td>$p_{22}$</td>
<td>Cello analysis</td>
</tr>
<tr>
<td>Brightness</td>
<td>${0, \ldots, 5}$</td>
<td>Ord.</td>
<td>Input</td>
<td>$p_{23}$</td>
<td>Cello analysis</td>
</tr>
<tr>
<td>Noisiness</td>
<td>${0, \ldots, 5}$</td>
<td>Ord.</td>
<td>Input</td>
<td>$p_{24}$</td>
<td>Cello analysis</td>
</tr>
<tr>
<td>Onset Rate</td>
<td>${0, \ldots, 5}$</td>
<td>Ord.</td>
<td>Input</td>
<td>$p_{25}$</td>
<td>Cello analysis</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Type</th>
<th>I/O</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
</table>

Table 8.7: The music system variables of the *Cello Improvisation* system. In the Type column, *categorical* is abbreviated by Cat. and *Ordinal* is abbreviated by Ord. The domains of variables corresponding to Ableton Live tracks include the clip numbers (from 0 upwards) and the value -2 which indicates that no clip is playing in the track.

the previous bar. Finally, each of these five values was normalized and quantized to an integer between 0 and 5 and these five integers formed the music system input variables, $p_{21}, \ldots, p_{25}$ (see Table 8.7). This coarse quantization was used with the aim of increasing the likelihood that rules would be discovered describing dependencies between the cello player’s performance and the agent’s output. Like the feature extractor of the Automato system, the one used here is based on ideas used in a number of contemporary interactive music systems reported in the literature (see the overview of common design techniques for sub-symbolic feature extractors in Section 1.3.4), and we discuss possible alternatives in Section 8.3.
Training Data and Agent Design

As with some of the previously discussed cases, certain logistic constraints interfered with the agent design process. Bown’s schedule conflicted with that of the cellist so it was not possible to record a live performance as a training example. Instead, the cellist provided a recording of a solo performance of approximately six minutes in length, demonstrating the style in which he envisaged playing during the performance. To create a training example, Bown played this recording back in place of a live cellist while performing with the Ableton Live set. In the resulting training data Bown’s performance continued after the cello recording had ended, so that for approximately half of the training example, the five cello analysis values did not change.

As with the agents produced during the Hacktogether, we were working in a rapid development context (the cello recording was supplied not long before the concert) so despite this incomplete training example, we decided to continue with the agent design process. As with the Automato system, our primary goal was to introduce some responsiveness into the agent’s behaviour without having particular responses in mind. Thus, our aim was to create a learning configuration that would generate a sufficiently responsive model using this training example. In the event that the data made this too difficult, we had recourse to truncate the training example to remove the portion without cello analysis values.

The single training example recorded is shown in Figure 8.7. The ending of the recorded cello performance can be clearly seen in the top five rows \((p_{21}, \ldots, p_{25})\). The entire performance is thirteen minutes in length (345 bars in a time signature of 4/4 at 106 beats per minute) and the cello performance ends after approximately six minutes (154 bars). In the figure, looped samples (those that continue for a number of consecutive bars) can be clearly seen in contrast with samples that were not looped (those that appear for a single bar and then stop). In addition, the use of the ‘scenes’ feature of Ableton Live is evident. As described in Section 2.1.1, this feature is a convenient method for triggering multiple clips simultaneously. For instance, if
the fourth ‘scene’ is triggered, then the fourth clip in every track will play (those tracks that do not contain a clip in the fourth position will stop). This accounts for the vertical patterns of particular colours in the figure (e.g. purple corresponds to the value 7, so the prominent purple-coloured vertical strip indicates that the 7th scene was triggered, causing all tracks to either play the 7th clip, or stop playing altogether).

Broadly, the Ableton Live tracks in the Cello Improvisation set can be divided into three types:

1. **Tonal Instruments**: Seven tracks contained prominent tonal material. These were the four organ tracks \(p_{1}, \ldots, p_{4}\), the two piano tracks \(p_{5} \text{ and } p_{6}\) and the tuba track \(p_{14}\).

2. **Metallic Percussive Sounds**: The four ‘rust’ tracks \(p_{8}, \ldots, p_{11}\).

3. **Bass**: The bass track \(p_{7}\).

4. **Incidental and Atmospheric Sounds**: The remaining tracks contained various other sound material.

Note that this kind of conceptual grouping of musical material was reflected in the way that it was laid out in the Ableton Live set itself. Furthermore, Ableton Live encourages this sort of organisation with its facilities for grouping similar tracks, that is, gathering adjacent tracks into a group so that effects and other operations can be applied to all tracks in the group at once. It also corresponds directly to our notion of ‘equivalence classes’ of musical material (introduced in Section 5.4.6) which are sets of material that are interchangeable in some way. This hints at the potential of a user interface innovation whereby such groupings could be used to inform the automatic creation of effective learning configurations.

Returning to the agent design process, the groupings above are useful for articulating the following stylistic requirements for the agent:
Figure 8.7: The training data for the Cello Improvisation system. There are 345 decision points in total. The Cello performance ends after 154 decision points (see text). The vertical stripes of particular colours result from the use of the 'scenes' feature of Ableton Live (again, see text).
• The performance should begin with the metallic percussive sounds with no prominent tonal material.

• Once the tonal material is introduced, it should remain throughout, forming different combinations and juxtapositions.

• The incidental and atmospheric sounds should be used intermittently during the performance.

As with the design of the Many Piano Chops agent, a prominent strategy during this design process was to try to infer from the training example what custom variables, VMMs and rule groups would be required to achieve these requirements.

The learning configuration we arrived at is shown in Table 8.8. The highest-priority variable is Tonal Playing, a binary-valued custom variable that determines whether one or more of the tonal instruments are playing. It is modelled using a 5th-order VMM. This provides a guarantee that at the beginning no tonal instruments will play but once they begin to play, they will continue throughout (this is true in the training example). The piano tracks have the next highest priorities and they are also modelled using 5th-order VMMs. This means that once the Tonal Playing variable takes a value of 1, the piano tracks will be almost entirely controlled by their VMMs, since they have a higher priority than all other variables (the piano tracks may turn off when the Tonal Playing variable is 1; this will create a requirement that at least one of the other tonal instruments switches on). The only instance in which a value generated by one of the piano tracks’ VMMs would be rejected is if it conflicts through the rules describing their dependency on the cello analysis values (this is discussed further below).

There are two other custom variables similar to Tonal Playing, governing the ‘rust’ tracks (Metal Playing) and the organ tracks (Organs Playing). These were created with the aim of allowing the musical model (i.e. the agent) to generalise effectively from the training data. In other words, the aim was to emulate the example in the way that these classes of instruments are introduced and removed, but not in the details
8.2. Case Studies

Table 8.8: The learning configuration for the Cello Improvisation musical agent. The variables are listed in priority order (see Section 5.4.5) with the highest-priority variable at the top. For custom variables, the labels, formulae (see Section 5.4.6, particularly Table 5.8 for an explanation of the notation) and VMM orders are shown, while for other (‘non-custom’) variables the formulae are replaced by the appropriate symbols. The VMM order is shown as ‘None’ when the value of a variable is completely determined by the values of higher-priority variables. For each rule group, the the minimum support, $S_{\text{min}}$, the minimum confidence, $C_{\text{min}}$, and the maximum itemset size, $\#_{\text{max}}$, (see Section 5.4.4) are shown; along with a list of the variables included in the group.
of the clips playing in individual tracks.

To give an example of the type of reasoning used to determine candidate learning configurations, it is interesting to consider the interaction between the Tonal Playing custom variable; the Piano 1 and Piano 2 variables; the Organs Playing custom variable; and the Tuba variable. If Tonal Playing is 0, then the piano tracks must not play (i.e. Piano 1 and Piano 2 must have values less than or equal to -1; they must be ‘off’), and in addition, Organs Playing must be 0 (since the organ tracks are a subset of the tonal instruments) and the tuba track must not play either. However, if Tonal Playing is 1, then any or all of the tracks governed by Piano 1, Piano 2, Organs Playing and Tuba may be active. Finally, the tuba track will be forced to play on occasions when Tonal Playing is 1, both of the piano tracks are off and Organs Playing is 0. This is because the Tuba variable has the lowest priority of those variables that interact with the Tonal Playing variable (which has the highest priority). In the above description, the potential effects of rules involving incoming cello analysis values are ignored.

Following the first four variables, the Rust Patter variable is modelled using a 3rd-order VMM. This order was chosen in order to better capture some of the changes exhibited in this track during approximately the first quarter of the example performance (again, see Figure 8.7). The remaining music system variables (Ableton Live tracks) were modelled using 1st-order VMMs so that there is potential for a lot of variety in the agent’s performance while maintaining some short-term continuity. Finally, the lowest-priority variable is the Pianos Playing custom variable (p29) which has the value 1 if one of the piano tracks is playing, and 0 otherwise. The VMM order of this variable is listed as ‘None’ because its value is completely determined by higher-priority variables (Piano 1 and Piano 2); this custom variable is relevant only to the rule groups.

Using the previously described iterative process whereby (i) the learnt rules were inspected and (ii) the rule groups were re-configured accordingly, we arrived at three rule groups (and associated parameters) which gave rise to rules that would introduce responsiveness into the agent’s behaviour. The first is the Groups and Cello
rule group which resulted in rules such as the following:

\[
\text{Cello Pitch} = 2 \text{ AND Cello Brightness} = 0 \implies \text{Organs Playing} = 1, \quad (8.1)
\]

\[
\text{Cello Noisiness} = 3 \text{ AND Pianos Playing} = 0 \implies \text{Organs Playing} = 0, \quad (8.2)
\]

\[
\text{Cello Noisiness} = 3 \text{ AND Pianos Playing} = 0 \implies \text{Metal Playing} = 1. \quad (8.3)
\]

These offer the potential for the agent’s decisions to be influenced by the cellist’s activities. The other two rule groups included selections of cello analysis variables and variables associated with Ableton Live tracks. Examples of rules resulting from these two rule groups include:

\[
\text{Cello Pitch} = 3 \text{ AND Cello Brightness} = 5 \implies \text{Tuba} = 0, \quad (8.4)
\]

\[
\text{Cello Brightness} = 0 \text{ AND Tuba} = 3 \implies \text{Samples 3} = 6, \quad (8.5)
\]

\[
\text{Cello Noisiness} = 2 \text{ AND Rust Patter} = 6 \implies \text{Rust 1} = 4. \quad (8.6)
\]

Thus, despite the non-ideal training example, it was possible to create a learning configuration such that rules were found to allow the cellist to influence the agent’s behaviour. We were able to audition the agent playing alongside the recording that had been provided by the cellist, in order to confirm that no catastrophic behaviours readily arose, such as abruptly ceasing to play, or playing with little or no variation.

**Agent Performance**

The *Cello Improvisation* system was presented at the *Computer Improvisers* concert, playing in tandem with the cellist who had supplied the recording for agent training. The performance lasted approximately 10 minutes. It was documented with an audio recording\(^8\).

Overall, the agent performed as intended, beginning without the tonal instruments and then introducing them and maintaining good variety and durations

---

\(^8\)Available at: soundcloud.com/runciblenoise/cello-and-agent-controlled.
throughout. There were clear musical changes as new elements were introduced and others were removed. On listening to a recording of the performance, Bown commented that it was ‘arguably better’ than an earlier duet in which he had played with the cellist using similar material (in which no ADTK agent was involved). Of course, this was not due to human-level musicianship on the part of the agent: Although the model did conform to our stylistic ideas, other factors played a significant role, such as the altered workflow that arose from using the ADTK; Bown noted that it ‘simplified what was going on.’

8.2.5 Backgammon

Context and System Description

The final agent design process described in this chapter was carried out for a system entitled Backgammon that was presented at a concert that was part of the programme of the 9th ACM Conference on Creativity and Cognition which took place in Sydney, Australia in June, 2013. The concert was held on June 19th. It was open to the public as well as to conference delegates.

The initial concept of Backgammon was to create a game-like interaction between a performer and a musical agent. To explore this, we obtained a database of Backgammon games, specifically the 17 games that were played in the final match of the 36th Backgammon World Championship between Takumitsu Suzuki and Fabio Gullotta in 2011. We then converted the record of each game into a format that could be used as a training example in the ADTK. This training data set comprising the 17 examples was then used as the basis on which to design a musical agent to control a specially arranged Ableton Live set in tandem with a live performer. The specifics of this process are given in the following.

In the game of Backgammon, a rectangular board is used on which 24 ‘positions’ are evenly distributed along two opposite sides (see Figure 8.8). Each position is
Figure 8.8: A Backgammon board ready for play. The board is oriented as it would be from the perspective of the player using the dark-coloured pieces. This player’s goal is to move all of his/her pieces anti-clockwise around the board to “bear them off” at the bottom right-hand corner.

marked with a triangle. The game is played by two players and each player begins with 15 playing pieces distributed in a specific way among the positions. The players take turns to roll two dice and move the counters around the board. The moves available to a player are constrained to a particular direction (i.e. clockwise or anti-clockwise around the board); by the numbers he/she has rolled on the dice; and by the arrangement of the counters.

A number of details of the mechanics of Backgammon are relevant to our creation of the musical agent and they are noted here (the following is not a complete description of the game). First, a player can only move one of his/her pieces to a position occupied by one or zero of the opponents pieces. If a player moves to a position occupied by one of his/her opponent’s pieces, then the opponent’s piece is moved to the ‘bar’ a line dividing the two halves of the board; thus at the end of a move, no position is occupied by pieces belonging to both players. Additionally, a single position can only be occupied by five or fewer pieces. Finally, each player’s objective is to move all of his/her pieces all the way around the board (clockwise for one player, anti-clockwise for the other) to ‘bear them off’ at the end (i.e. they are removed from play). The first player to bear off all of his/her pieces wins the game,
though in the context of a match, the game can be won under some circumstances before all 15 pieces have been borne off. From the above, it can be seen that there are 26 possible states for each piece since it can be in one of the 24 board positions and additionally it may be ‘on the bar’ or ‘borne off’ (also known as ‘home’).

We created an Ableton Live set according to a particular mapping from the configuration of pieces in a Backgammon game to musical material as follows. An Ableton Live contained two sets of 26 tracks (i.e. 52 tracks in total). The first set of 26 tracks corresponded to the 26 possible states in which a piece belonging to the first player (the live performer) may be found. Similarly, the second set of 26 tracks corresponded to the 26 possible states in which a piece belonging to the second player (the agent) may be found. The clip playing in each track corresponded to the number of a player’s pieces in the corresponding state. Thus for the 24 tracks corresponding to board positions, there were 5 clips (since usually, no more than about five pieces ever occupy a position) and for the tracks corresponding to ‘on the bar’ and ‘home’, there were 15 clips (since theoretically all 15 of the player’s pieces could occupy either of these states).

**Training Data and Agent Design**

Two of the 17 Backgammon games used as training examples are shown in Figure 8.9. Each column corresponds to the state of the game after a move by one or other of the players. Thus, the configuration of given player’s pieces only changes every two columns (i.e. every cycle in which each player moves once), except where the other player causes one of his/her pieces to be moved to the bar. The rows in the figure are labelled according to the player (P1 or P2) and the state (the 24 board positions, as well as ‘home’ and ‘bar’). The board positions are numbered from the point of view of the player, thus position 1 for player 1 is the same board position as position 24 for player 2; similarly position 2 for player 1 is the same as position 23 for player 2, and so on. To make this data compatible with Ableton Live, the value 0 was changed to -2 (i.e. when no pieces occupy a given state, the track corresponding to that state
is not playing), and all other values were reduced by 1 because the clips in Ableton Live are numbered from 0 (e.g. a state occupied by a single piece, corresponds to clip 0 playing in the relevant track).

In planning the performance, we did not envisage that the live musician would attempt to control his 26 Ableton Live tracks according to the rules of Backgammon. Thus, in designing the Backgammon agent, we aimed to capture just some of the dynamics of the game while placing no constraints on the musician. Note that we made no attempt to accurately capture the rules of Backgammon in an ADTK agent. We designed the agent with the following features of Backgammon games in mind:

1. When a board position is occupied by more than one of his/her opponent’s pieces, then a player cannot move any pieces there. For the agent, this means that when the musician is playing clip 1 or higher in a given track, then the corresponding track under the agent’s control must not be playing (e.g. if the musician is playing clip 2 in the track corresponding to \( p_2 \) (i.e. \( P1 \text{ Pos } 1 \)), then the track corresponding to \( p_{51} \) (i.e. \( P2 \text{ Pos } 24 \)) must be stopped.

2. The total number of pieces belonging to a player is always 15 (distributed among the 26 states) and the agent’s decisions should be correspondingly restricted. One implication of this, for example, is that if the agent plays the highest-numbered clip in the track corresponding to ‘home’ then no other clips can be playing since this corresponds to the game state in which all of the pieces belonging to player 2 are home.

3. The general structure of the individual states should be maintained, observing, for example that the number of pieces at a given position cannot change by more than two after a player’s move, and the number of pieces in the ‘home’ position must increase monotonically.

The learning configuration used for the Backgammon agent is shown in Table 8.9. Each of the 26 variables under the agent’s control are modelled using 2nd-order VMMs (for item 2 above). The order was chosen because of the way in which the
Figure 8.9: Part of the training data set for the Backgammon system. The lower 26 rows in each grid \((p_1, \ldots, p_{26})\) correspond to the 26 possible states in which pieces belonging to the first player (P1) may be found. Similarly, the upper 26 rows in each grid \((p_{27}, \ldots, p_{52})\) correspond to the 26 possible states in which pieces belonging to the second player (P2) may be found. In Example (a), the high values appearing towards end of the game (right hand side) in the P1 Home variable indicate that the game was won by the first player (P1), who also wins the game shown in (b), though this time less decisively (the second player gets many of his pieces home too).
training data was formatted; a player’s pieces generally move only every second column (see again, Figure 8.9). The priority ordering is explained below.

The custom variables with labels of the form $P1 \text{ Pos } X \text{ Clip } > 0$ are required for the rule groups with labels of the form $Interaction \ X$. Together, these custom variables and rule groups allow rules to be discovered that correspond to item 1 above. For examples, rules such as:

$$P1 \text{ Pos } 1 \text{ Clip } > 0 = 1 \implies P2 \text{ Pos } 24 = -2 \text{ and}$$

$$P1 \text{ Pos } 2 \text{ Clip } > 0 = 1 \implies P2 \text{ Pos } 23 = -2,$$

are discovered. These correspond to the Backgammon rule that at the end of a turn, only pieces belonging to one player can occupy any position. Ideally, 24 such rules would be found (one for each position), but in fact only 18 rules of this form are discovered because the training data set does not happen to contain examples for all positions.

It was not possible to accurately ensure that the total number of pieces controlled by a player in a Backgammon game is 15. An appropriate $Sum$ custom variable was included in early design iterations, however, it led to constraint satisfaction problem that took too long to convert into a binary decision diagram (BDD). It is not known how long the conversion might have taken as the process was stopped after one hour (the computation was run on a 2012 Apple Macbook Pro with a 2.6 GHz processor).

Since it was not possible to accurately constrain the number of notional ‘pieces’ to 15, an approximation of this constraint was introduced by adding two elements to the learning configuration. First, the custom variable labelled $Any \ Track \ Playing$ was added to ensure that the agent is always playing at least one clip. The domain of this variable, as learnt from the training data set, contains only the value 1, so merely including it ensures that at all times, at least one of the agent’s tracks is playing a clip. Second, 24 rule groups were added ($Home \ 1$, $Home \ 2$, etc.) to allow rules to be discovered corresponding to the fact that when there is a high number of pieces in
Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2 Home</td>
<td>(p_{27})</td>
<td>2</td>
</tr>
<tr>
<td>P2 Pos 1</td>
<td>(p_{28})</td>
<td>2</td>
</tr>
<tr>
<td>P2 Pos 2</td>
<td>(p_{29})</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>P2 Pos 24</td>
<td>(p_{51})</td>
<td>2</td>
</tr>
<tr>
<td>P2 Bar</td>
<td>(p_{52})</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 Pos 1 Clip &gt; 0</td>
<td>(p_{53} = \left[&gt;_{\text{ANY}} (0, p_2)\right])</td>
<td>None</td>
</tr>
<tr>
<td>P1 Pos 2 Clip &gt; 0</td>
<td>(p_{54} = \left[&gt;_{\text{ANY}} (0, p_3)\right])</td>
<td>None</td>
</tr>
<tr>
<td>P1 Pos 24 Clip &gt; 0</td>
<td>(p_{76} = \left[&gt;<em>{\text{ANY}} (0, p</em>{25})\right])</td>
<td>None</td>
</tr>
</tbody>
</table>

Any Track Playing \(p_{77} = \left[>_{\text{ANY}} (-1, p_{27}, \ldots, p_{52})\right]\) None

Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>({S_{\text{min}}, C_{\text{min}}, #_{\text{max}}})</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction 1</td>
<td>{0.001, 0.999, 2}</td>
<td>{p_2, p_{55}}</td>
</tr>
<tr>
<td>Interaction 2</td>
<td>{0.001, 0.999, 2}</td>
<td>{p_3, p_{54}}</td>
</tr>
<tr>
<td></td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>Interaction 24</td>
<td>{0.001, 0.999, 2}</td>
<td>{p_{25}, p_{27}}</td>
</tr>
<tr>
<td>Home 1</td>
<td>{0.001, 0.999, 2}</td>
<td>{p_{26}, p_{27}}</td>
</tr>
<tr>
<td>Home 2</td>
<td>{0.001, 0.999, 2}</td>
<td>{p_{26}, p_{28}}</td>
</tr>
<tr>
<td></td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>Home 24</td>
<td>{0.001, 0.999, 2}</td>
<td>{p_{26}, p_{51}}</td>
</tr>
</tbody>
</table>

Table 8.9: The learning configuration for the Backgammon musical agent. The variables are listed in priority order (see Section 5.4.5) with the highest-priority variable at the top. For custom variables, the labels, formulae (see Section 5.4.6, particularly Table 5.8 for an explanation of the notation) and VMM orders are shown, while for other (‘non-custom’) variables the formulae are replaced by the appropriate symbols. The VMM order is shown as ‘None’ when the value of a variable is completely determined by the values of higher-priority variables. For each rule group, the minimum support, \(S_{\text{min}}\), the minimum confidence, \(C_{\text{min}}\), and the maximum itemset size, \(#_{\text{max}}\), (see Section 5.4.4) are shown; along with a list of the variables included in the group.
the ‘home’ state, few pieces can be elsewhere on the board (because most of them have been borne off). These rule groups give rise to rules such as:

\[ P2 \text{ Home} = 14 \implies P2 \text{ Pos 13} = -2 \] and

\[ P2 \text{ Home} = 8 \implies P2 \text{ Pos 12} = -2. \]

Thus, they capture features of the data that arise from the fact that player 2 only has 15 pieces. These rules provide the reason for giving \( P2 \text{ Home} \) the highest priority, since its value constrains those of other variables.

**Agent Performance**

The *Backgammon* system was presented with Bown as laptop musician. The performance lasted approximately 11 minutes and was documented both with an audio recording, and a log of the agent’s decisions. *Backgammon* was intended as an abstract dynamical system, thus, unlike all of the other agents reported in this thesis, it was not created with particular stylistic requirements in mind. Instead, it provides an example of the ADTK being used for agent design in another creative context.

Having no stylistic requirements, it is difficult to make comments on the agent’s musical performance. However, we did feel that the combination of the agent and material did not effectively convey a game-like interaction; there was no clear differentiation between the material being controlled by the agent and that being controlled by the musician, and there was no sense of an action on the part of one being responded to by the other. Thus, this somewhat madcap system, while conceptually interesting, did not fully achieve our goals.

We note, however, that this agent is significant from the perspective of computational performance. It is one of the most complex agents reported in this thesis, using 103 variables and custom variables and requiring the values of 52 variables to be set at each decision point. The recorded log file showed that during the performance,

---

10Available at: soundcloud.com/runciblenoise/backgammon-performance.
all update times took between 1 ms and 19 ms.

8.3 Discussion

8.3.1 The Use of the ADTK in Bricoleur-Style System Development

One striking observation of the agent design processes reported in this chapter is the speed with which agents could be created. This is especially important in the rapid development contexts associated with bricoleur-style system development. Three of the five agent designs were carried out with severe time constraints. For the Automato and Many Piano Chops systems, the example performances were not ready until close to the time of the public performances. Each of these agents was created in approximately one hour. The Cello Improvisation agent was created in less than two hours. While not all of our musical goals were achieved in every case, this does give an indication of the ease with which an interactive system can be created, and the speed with which lower level feature extractors and generators can be combined into a single coherent system. In light of the fact that there are no other tools for designing arrangement-level decision makers that can be integrated readily into standard music production platforms, we regard this as an important and exciting result.

Also relevant to the use of the ADTK in bricoleur-style development is the variety of types of musical agents to which we have demonstrated its applicability. Using the terminology introduced in Chapter 1, the Automato and Cello Improvisation systems are player-paradigm interactive systems, that is, ones designed to improvise alongside a musician playing an acoustic instrument. Each of these systems resulted from a deliberate design process carried out with specific stylistic requirements in mind. This contrasts with the Backgammon system which was essentially a Live Algorithm [28] created very much in the spirit that Perkis described as ‘want what you get’ [36]; by using data from Backgammon games and choosing features of the
data to learn without paying heed to their musicality (or lack thereof), we essentially removed ourselves from the design process with respect to the agent’s musical behaviour.

Like the three systems mentioned so far, the Many Piano Chops system played a role in improvised performance though it was not interactive. The use of a non-interactive system in improvised performance may seem somewhat ill-conceived. However, it’s dual purpose was to provide a backdrop against which the musicians could improvise and to delineate a section of the improvisation. The former recalls Dean’s comment that

‘A generative algorithm might also be conceived as poorly interactive, but providing a sound environment in which other sound is improvised. Unpredictability in the algorithmic output is also a potential positive contributor to its improvisatory utility.’ [67, pp. 73-74]

Thus, a generative system may play a significant role in the context of a musical improvisation, and this is what we observed.

The Isidores system was a more standard generative system. Despite being implemented outside of Ableton Live, it was very similar in concept to the style emulation systems developed in the previous chapter. From an artistic perspective, the creation of a system like this instead of a non-generative composition can be motivated using Drummond’s words which were written with interactive systems in mind, but which can equally apply to generative ones:

‘Just as a sculpture can change appearance with different perspectives and lighting conditions, yet a sense of its unique identity is still maintained, so too an interactive sound installation or performance may well sound different with subsequent experiences of the work, but still be recognisable as the same piece.’ [71]

Finally, the Isidores and Automato systems demonstrate the use of ADTK-designed agents in systems with decision making processes operating on two time scales (this
is common in the systems surveyed in Section 1.3). In each, the arrangement-level musical decision maker (i.e. the musical agent) makes choices on a longer time scale (once per bar in the *Isidores* system, and once per phrase in the *Automato* system) and the generator module makes choices on a shorter time scale (choosing individual notes, rhythms and other low level sound parameters). The generators of the other three systems were static sequences stored in Ableton Live clips.

### 8.3.2 The Agent Design Workflow

In four of the five cases reported above (all except the *Backgammon* system), the workflow used to create an interactive or generative system was as follows:

1. Create the feature extractor and generator modules.
2. Record one or more example performances.
3. Explicitly identify requirements for the agent’s behaviour.
4. Iteratively design a learning configuration that leads to a model that satisfies these requirements.

The first two steps of this workflow are imposed in situations where style emulation is required. They are not specific to the ADTK, since examples must be performed that exhibit the style and to do that, the underlying components of the system required to record the examples (the feature extractor and generator) must be complete.

After the recording of example performances, the third phase in the workflow requires the identification and articulation of stylistic requirements. In the cases above, we gave examples of difficulties related to this, particularly in identifying

- very rudimentary requirements (e.g. ‘there should always be something playing’);
- ‘negative’ requirements (e.g. ‘this instrument should *not* play too much’); and
• requirements relating to musical actions that may arise only very infrequently (e.g. particular unlikely combinations of parameters that lead to catastrophic results).

Of course, in situations where there is time to conduct a meticulous design process, these requirements may be discovered by auditioning intermediate agent designs. However, particularly in collaborative processes in which the auditioning of agents—especially interactive ones—may be severely curtailed, there is a greater need to determine them through informed inspection of the training examples and the learning configuration with a clear understanding of the desired behaviour. This issue is related to the broader one of ensuring that the agent’s model of music is well understood by the user and this is discussed in detail in Chapter 10. Here, we note that these difficulties support the position we have taken in this thesis that the current implementation of the ADTK, while successfully removing the need for expertise in conventional computer programming, would benefit from a higher-level design interface which would provide methods for fulfilling commonly arising (and sometimes easily overlooked) stylistic requirements, and which might even provide suggestions as to when such methods should be employed.

Finally, we note that the Backgammon system is an exception to the above discussion, since it did not require style emulation. We began with the training data and designed a system in order to utilise it; the ADTK was being used simply to create an abstract dynamical system. In this context the workflow above is not imposed since examples may be taken from earlier work, or completely abstract data. This paradigm has exciting creative prospects and is very much in line with the notion of behavioural objects [32] which are transferable software entities embodying particular dynamical behaviours for use in computer music systems. In addition, from a usability perspective, this suggests improvements to the software, since it currently does not have features such that inside Ableton Live, it is easy to reconfigure the elements of the Ableton Live set that are being controlled by a given variable.


8.3.3 Designing Interactive Agents

In this section, we consider the three interactive agents reported in this chapter (Automato, Cello Improvisation and Backgammon). These are the first demonstrations presented in this thesis of the ADTK being used to design interactive agents. In Section 1.5, we gave an overview of the term ‘interactive’ as it applies in the field of computer music. In particular, we noted that in the computer music literature the term is used to describe a system that strikes a balance between being simply ‘reactive’ (like an instrument) at one extreme, and having a mapping from input to output that is completely incomprehensible (i.e. the agent is effectively unresponsive), at the other.

The nature of the responsiveness of ADTK-designed agents is deterministic. That is, when a particular set of input values are supplied to an interactive agent, the space of possible values of the output variables is modified in a deterministic way. A particular set of values is chosen randomly from that space, with a probability distribution governed by the particular model being used (i.e. the VMMs, priority ordering, custom variables and rules) and the variable histories used by the VMMs. Thus, whether an ADTK-designed agent is reactive, unresponsive, or somewhere in between, is determined by the way in which the space of output values available to the agent is affected by the input values. If the input values reduce the space of output values by too much, the agent will tend towards reactivity, whereas if they have little effect on the space of available output values then the agent is likely to appear unresponsive. Furthermore, this balance is not independent of the generators and the type of musical material that they produce; if the input values tightly control a very prominent generator, then this might lead to a perception that the agent is reactive whereas if they tightly control a generator that only has a minor effect on the sound output by the agent then this may not be the case.

The search for a balance between reactivity and unresponsiveness was evident in design of the Cello Improvisation agent. The learning configuration at which we arrived, gave rise to rules constituting a mix between ones that produce particular
responses (e.g. Rule 8.5 in which the Samples 3 track must to play a particular clip) and more broad responses (e.g. Rule 8.1 in which any of the four organ tracks must play, but it is not specified which one or which clip). In contrast, many of the rules of the Backgammon agent were instrument-like (i.e. reactive) since when the performer chose certain clips among those under his control, this would deterministically stop certain tracks being played by the agent (corresponding to the rules in the game of Backgammon whereby only a single player’s pieces can occupy a certain position on the board). However, since there were a large number of tracks playing and since the material was similar across many tracks, this may not have been perceptible to the audience (especially when it was not clear to them which aspects of the music the performer controlled, and which the agent controlled).

In addition to the rules that form the agent’s model of musical performance, the feature extractor is an important component of an interactive system. For strongly prescribed situations in which the agent is required to emulate the behaviour of a laptop musician playing alongside an acoustic instrumentalist, for example, then a feature-extractor is required that is proven to produce descriptions of the acoustic musician’s activities that correspond veridically to the features that influence the laptop musician’s decisions. In the cases of the two systems that played alongside acoustic instrumentalists (Automato and Cello Improvisation), we used only crude feature extractors that calculated coarsely quantized statistics of perceptually informed attributes. We do not claim that these provided anything more than an abstract representation of the acoustic environment with which to influence the agent’s decision making (though we do note that the feature extractors were informed by some of those used in various published systems; see Section 1.3.4). In the future, we plan to investigate state of the art feature extractors that produce higher-level descriptions of the activities of the acoustic instrumentalist, in the spirit of the style classifiers described in [65] and [98]. For example, in the former work a real-time performance system was developed that classified an instrumentalist’s activity according to descriptors such as ‘lyrical’ and ‘pointillistic.’
8.3.4 Reflections on the Agent Design Process

Most striking, and perhaps surprising, about our experience of the agent design process is that auditioning the agent is a fatiguing process, and moreover, this cannot be accounted for by the occasional logistic difficulties that arose. It may in part be because the agent is designed for arrangement-level decision making, and therefore significant time and concentration are required to gain an understanding of the space of possible performances. During the agent designs reported in this chapter, we frequently employed inspection of the learnt rules as a shortcut to auditioning the agent; in many cases this was sufficient to discover whether a particular change to a rule group resulted in a promising model. This leads to the prospect that there are other ways, perhaps more effective ones, to efficiently give the user an understanding of the learnt model. This is discussed in greater detail Chapter 10 in the context of interactive machine learning.

In addition to employing shortcuts to auditioning, we endeavoured to minimise the number of design iterations, primarily by attempting to infer a promising learning configuration at the outset. This involved using knowledge of (i) the desired outcomes, (ii) the training data, and (iii) the learning mechanisms underlying the ADTK. Minor examples of the types of reasoning required for this were given throughout our accounts in this chapter, with a more extensive one being provided in our description of the Cello Improvisation agent, concerning the interaction between custom variables for a given priority ordering. Clearly, this type of analytical reasoning is somewhat similar to that required for computer programming and as such, insofar as it is required, it provides further support for our view that a higher-level design interface would increase the accessibility of the software. However, we note that this sort of reasoning was primarily needed to elicit very specific behaviours and this may not always be required.

A final aspect of the agent design process that we note here arose during the design of the Backgammon agent. As mentioned in the account above, we encountered a number of agent designs that gave rise to constraint satisfaction problems that
could not be quickly converted to BDDs. This slowed the design process down but perhaps more importantly it gave rise to apprehension about the time it would take for an agent to load. This impeded the free exploration of different designs; when an agent loaded quickly, we experienced a cognitive barrier to making further modifications lest they should result in an agent that could not be loaded. This was partly due to the fact that it was required to quit the software in order to stop the agent loading process, and so it could easily be alleviated to a great extent by the addition of a progress bar and an appropriate ‘cancel’ button. In addition, we do not expect this problem to be particularly widespread, since the size of the Backgammon system was relatively ambitious; we speculate that 52 tracks of audio is unusual in live performance, and it is the commensurately large number of variables that gave rise to the problematic agent designs. Nevertheless, the provision of alternative methods for computing the real-time decision making (i.e. alternative implementations of the Fast Performer module of the ADTK) may also be a worthwhile avenue for future research.

8.4 Conclusion

We have illustrated the application of the ADTK to the design of generative and interactive music systems in a variety of contexts. With reference to research question III-(iv) (see Section 1.6.2), the speed with which it allows musical agents to be designed makes it highly integrable into the workflows of contemporary musicians engaged in bricoleur-style development. However, we did identify design challenges such as the difficulty of auditioning agents sufficiently, that should be addressed in future research.

We acknowledge that the typical participant in a collaborative system design workshop is a musician-programmer and is therefore considerably different from the envisaged non-programming end user of the ADTK. However, when used inside Ableton Live, the need for conventional programming is removed (no programming
was required for the Backgammon agent), since the ability to use the actions of a computer musician (feature extractor) to influence the decisions of a musical agent in its control of Ableton Live clips and audio processing (generator) is built-in to this set of tools. This motivates the future development of alternative feature extractors to be used inside Ableton Live and perhaps based on those used in the Automato and Cello Improvisation agents, so that the combination of Ableton Live and the ADTK can be more easily used to build systems that interact with acoustic musicians.

In addition we have studied the agent design process using the ADTK, and characterised its use in collaborative contexts and in designing interactive agents. This discussion leads more broadly to the characterisation of the ADTK both as a tool for supporting creative work (a creativity support tool) and in the context of interactive machine learning. These topics underly much of the discussion in Chapter 10. However, in the following we study the usability of the ADTK more closely.
Chapter 9

A Study of the ADTK from the Perspective of End-User Programming

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Analysis of the ADTK Notation</td>
<td>327</td>
</tr>
<tr>
<td>9.1.1</td>
<td>The Cognitive Dimensions of Notations</td>
<td>327</td>
</tr>
<tr>
<td>9.1.2</td>
<td>A Cognitive Dimensions Analysis of the ADTK</td>
<td>329</td>
</tr>
<tr>
<td>9.2</td>
<td>A Preliminary Usability Study of the ADTK</td>
<td>333</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Method</td>
<td>333</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Results</td>
<td>335</td>
</tr>
<tr>
<td>9.3</td>
<td>Discussion</td>
<td>347</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Attentional Investment</td>
<td>347</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Creation of the Learning Configuration</td>
<td>348</td>
</tr>
<tr>
<td>9.3.3</td>
<td>Evaluating the Agent</td>
<td>350</td>
</tr>
<tr>
<td>9.4</td>
<td>Conclusion</td>
<td>350</td>
</tr>
</tbody>
</table>
End-user programming (EUP) relates to the extension of existing software, or the creation of new software for personal use, rather than for use by others\footnote{Related to EUP is end-user development (EUD) which encompasses EUP, but also includes ‘the entire software development lifecycle, including modifying and extending the software, not just the “create” phase’ \cite{learned:2001}. EUD can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artefact \cite{learned:2001}.} \cite{van2000}. Well known examples of EUP include the creation of macros to enhance the usability or extend the functionality of an application, and the use of spreadsheets to perform custom calculations. While EUP is not specific to the creation or extension of software by ‘novices’ or ‘non-programmers’, research has frequently focussed on such individuals. This has lead to the exploration of a wide range of interaction styles through which EUP might be performed (see \cite{learned:2001} for a review), including text-based programming, visual programming languages and programming by example (PBE). Regardless of the interaction style, a number of challenges are commonly encountered, such as the choice of notation manipulated by the user to create the program, the level of abstraction allowed (and required) and by what mechanisms the the program can be debugged. Another challenge, particularly in the case of PBE, is how to represent the program to the user once it has been created (or learnt by the system).

The ADKT can clearly be seen as an EUP tool for programming the behaviours of musical agents and many of the challenges discussed in the previous chapter can be cast in terms of the issues mentioned above. The notation used by the ADTK is that of variables, custom variables, VMMs and rule groups, and the use of abstraction, particularly in the form of custom variables, is fundamental to agent design. The representation of the inferred program (the agent) to the user is currently limited to displaying the rules discovered and auditioning the agent. Finally, debugging an agent requires careful reasoning about the interactions between the different variables, VMMs, rule discovery and their relation to the training examples themselves.

In this chapter, we study the usability of the ADTK using techniques from the
field of EUP (see research question III-(v), Section 1.6.2). The study comprises two parts. First, in Section 9.1 we present an analysis of the notation used in the ADTK using the Cognitive Dimensions of Notations framework (see below). Next, we report (Section 9.2) on a preliminary usability study of the software in which a single user was tasked with designing an agent. We then discuss the results of the two parts from the perspective of EUP (Section 9.3).

9.1 Analysis of the ADTK Notation

9.1.1 The Cognitive Dimensions of Notations

The Cognitive Dimensions of Notations [22] (abbreviated CDs) is a conceptual framework for the analysis of the usability of information artefacts, that is, ‘tools we use to store, manipulate, and display information’ [89]. Information artefacts generally comprise a notation, the visual marks used to represent information structures; an environment in which the notation can be modified; and a medium on which the notation is presented, such as a computer screen. CDs have been applied in many contexts of great relevance to EUP, such as programming languages [54] and visual programming languages [90]. The CDs framework was introduced as a set of discussion tools for designers, but it has also been used in the form of a questionnaire [23] by which users could report their experiences with a piece of software.

Central to the CDs framework is a set of Cognitive Dimensions which are descriptors that can be used to characterise a notation and its environment. The CDs characterisation can then be used to evaluate the software with respect to various activities in which a user might engage. The Cognitive Dimensions are as follows:

- **Viscosity**: The resistance to change (modification of information).

- **Visibility and Juxtaposability**: The ease with which components can be viewed and compared.
• **Premature Commitment:** The extent to which there are constraints on the order of doing things that may force the user to make choices before all the necessary information is available.

• **Hidden Dependencies:** Important links between entities that are not visible.

• **Role Expressiveness:** The ease with which purpose of an entity can be inferred.

• **Error-Proneness:** The extent to which mistakes are easily made.

• **Abstraction:** The availability and types of abstraction mechanisms, and the requirements placed on the user with regard to them.

• **Secondary Notation:** The availability of extra information other than the formal syntax.

• **Closeness of Mapping:** The extent to which the notation is related to the result.

• **Consistency:** The extent to which similar semantics are expressed in similar syntactic forms.

• **Diffuseness:** The verbosity of the notation.

• **Hard Mental Operations:** High demands on cognitive resources and working memory.

• **Provisionality:** The degree of commitment to actions or marks; the extent to which the environment allow the user use placeholders, or other imprecise markings.

• **Progressive Evaluation:** The extent to which the work can be evaluated at any time.

In addition, the prototypical activities with respect to which a piece of software can be evaluated, are [22]:

• **Incrementation**: The incremental addition of information.

• **Transcription**: The transcription of information from another notation into that of the software.

• **Modification**: Modification of information structures within the software.

• **Exploratory Design**: Sketching and testing possible solutions when the final product cannot be planned in advance.

• **Searching**: Searching for known targets (e.g. a particular option or function in a piece of software).

• **Exploratory Understanding**: Working to discover structure in the information.

The CDs authors emphasise that no cognitive dimension is ‘good’ or ‘bad’ except with respect to an activity. For example, high viscosity is not a problem for the incrementation activity but it is for the modification activity. Of the six activities, exploratory design is most relevant to designing agents with the ADTK. It encompasses many of the other activities and thus from a design perspective, it the most demanding activity to support [22] (see below).

### 9.1.2 A Cognitive Dimensions Analysis of the ADTK

In the next section, we analyse the ADTK using the CDs framework. For completeness, we address each of the fourteen Cognitive Dimensions and then consider the analysis in the context of exploratory design. However, we note that the user interface of the ADTK was implemented as a prototype to support the studies described in the previous two chapters. Even a cursory glance through Appendix A reveals numerous obvious design flaws such as the lack of useful secondary notations (no comments are possible, for example), the difficulty modifying examples (viscosity), and the inability to juxtapose examples, custom variables or rule groups (visibility and juxtaposability). Therefore, in the following, while we consider both the environment (the ADTK) and the underlying notation (examples, variables, custom
variables, VMMS, rule groups and rules), we are most interested in the issues originating in the notation, rather than the rudimentary graphical user interface that we implemented.

**Viscosity:** The ADTK is quite viscous in a number of ways, particularly in relation to custom variables. For example, to create a custom variable representing the number of tracks playing in Ableton Live, a very high number of individual actions are required. In addition, individual values in examples can be modified, but no ‘broad-brush’ modifications can be made without undertaking painstaking work.

**Visibility and Juxtaposability:** In the current ADTK interface, there are areas where visibility and juxtaposability might be improved. For example, it would certainly be an improvement to display examples as recorded performances are displayed in Ableton Live (i.e. using blocks of colour), rather than as tables of numbers. However, the most glaring issue with respect to visibility and juxtaposability concerns the representation of agents. Apart from displaying the learnt rules, we have found no way to visualise an agent. Currently, the only way to ‘view’ an agent is by listening to its performances. This makes it difficult to get a complete understanding of an agent’s behaviour and also to compare agents with one another.

**Premature Commitment:** The ADTK does require musical material and examples to be created before the agent design is begun. However, during the agent design process, there are few constraints on the order in which things are carried out.

**Hidden Dependencies:** This is perhaps the single most significant descriptor in this analysis of the ADTK. When designing an agent, there can frequently be complex interactions between the example data, variables, custom variables, VMMS and rules, mediated in part by the variable priorities. For example, a BLOCK variable can be used in a straightforward way to enforce hypermetrical structure. However, another variable can disrupt this if it has a higher priority and it interacts with the BLOCK variable through discovered rules. Moreover, whether such rules exist depends on the selected rule groups and the example data.
9.1. Analysis of the ADTK Notation

Role Expressiveness: While the function of a greater than custom variable is likely to be readily inferred by a computer programmer or mathematician, this may not be true of an average computer music practitioner. Its most common use in the agents presented over the previous two chapters was to represent whether or not a particular track in Ableton Live was playing. The term ‘greater than’ does not hint at this at all. Similarly, the most common use of the sum custom variable was to count the number of Ableton Live tracks playing and this purpose would not likely be inferred by a user without explicit instruction. The mathematical terminology therefore, while perfectly general, does little to aid the user.

Error-Proneness: Due primarily to the above-mentioned pervasiveness of hidden dependencies in the ADTK, errors are very easily made.

Abstraction: The ADTK has an abstraction barrier \[89\], meaning that there are abstractions that must be learnt by users before they can use the system. Additionally, it is abstraction-hungry \[89\], requiring the user to define abstractions before making any significant progress. In both cases, the abstractions include all of the fundamental elements of the learning configuration: VMMs, custom variables (especially), and rule groups.

Secondary Notation: The ADTK allows users to freely choose names for custom variables and rule groups, but no other descriptive annotations are possible. There are no redundant notations in place.

Closeness of Mapping: In the ADTK, there is clearly a wide gulf between the learning configuration and the resulting musical behaviour of an agent. This is due to many of the factors already mentioned, such as the presence of hidden dependencies and lack of role-expressiveness. Indeed, the notation includes no terms whatsoever that are related to music, music performance or Ableton Live.

Consistency: The notation used in the ADTK, while suffering from the issues already mentioned, is not extensive. It is relatively consistent throughout.

Diffuseness: The ADTK is relatively diffuse, using full words in most cases, rather than symbols. However, the limited extent of the notation may reduce the
significance of this descriptor.

**Hard Mental Operations:** Due to the number of abstractions and the high likelihood of hidden dependencies, the ADTK is likely to place significant demand on a user’s cognitive resources and working memory. This is primarily due to the need to carefully think through the possible outcomes of combining various custom variables and modelling options, with cognisance of the particular example data being used. While it may be possible to forego this sort of thinking to some extent in favour of a more trial-and-error approach, effective debugging would still require careful analysis.

** Provisionality:** The ADTK does not allow the user to create placeholders or other imprecise markings.

**Progressive Evaluation:** Agents can be saved and auditioned at any point during the design process.

**Exploratory Design**

As noted above, exploratory design is the most demanding activity to support, requiring in particular [22, 89]

- low viscosity,
- few hidden dependencies,
- no premature commitment,
- a low abstraction barrier and no abstraction hunger (see above),
- no secondary notation,
- high visibility and juxtaposability, and
- high role-expressiveness.

Thus, from the analysis above the key issues with regard to the notation of the ADTK (rather than the interface, specifically) relate to hidden dependencies, abstractions,
visibility and juxtaposability, and role-expressiveness. Though not specifically mentioned by the CDs authors, the lack of closeness of mapping and the necessity for hard mental operations are surely significant issues as well.

9.2 A Preliminary Usability Study of the ADTK

In this section, we give details of a preliminary usability study of the ADTK. Despite the fact that the ADTK, as presented in this thesis, has only a prototype interface, we conducted a usability study to provide empirical support for the results of the previous section (note that the CDs analysis was carried out before the usability study was conducted). In particular, we were interested to see whether a user would be able to take advantage of custom variables to achieve his/her agent design goals. For this preliminary work, a single participant was recruited from the mailing list of the Australasian Computer Music Association. The participant was an active computer music practitioner and a trained sound engineer.

9.2.1 Method

To begin, the participant was given a tutorial on the ADTK by the experimenter. The use of custom variables and rule groups were outlined, and instruction was provided on the use of the ADTK interface. Moreover, a selection of the techniques arrived at in Chapter 7 were explained, such as the use of a SUM variable to model the dynamic structure (see Section 7.3).

The participant was given details of a scenario in which he/she was tasked with providing to a fictional client, a musical agent that would be used to generate music during certain periods in a computer game while the player waited for content to load. The agent was required to generate music using a certain pre-defined set of musical material and in a style exemplified in a set of provided examples, but the music should have as much variety as possible within these constraints to prevent it becoming too predictable after repeated listening.
The participant was then provided with an Ableton Live set. The set comprised eight tracks with one, two or three clips on each track. All clips were either one, two, three or four bars long. In addition, he/she was provided with four example performances\footnote{An audio rendering of one example performance, as well as performances by a selection of agents designed by the participant, are available at: am-process.org/thesis-examples.} with durations 2’38”, 2’21”, 2’46” and 2’08”.

The example performances constituted ambient electronic music. The primary melodic instruments were a bass instrument and a lead melody instrument. The clips played by each of these instruments were all four bars long and followed a particular repeating chord progression. In the example performances, when these instruments were used together, they were played in synchrony so that the four-bar chord progressions coincided with one another. Thus, there was a clear four-bar hypermetrical structure in the sequencing of the lead and bass instruments. A third melodic instrument was tonally neutral with respect to the chord progression. The remaining five instruments were percussive. These, along with the three melodic instruments were layered in various ways during each performance. Instruments were usually introduced and removed after multiples of two or four bars. Both buildups and breakdowns were used in the example performances. Note that to successfully replicate hypermetrical structure and dynamic changes using the ADTK, effective use of custom variables is required.

The participant was asked to go about constructing a musical agent according to the brief while thinking aloud, that is, verbalising his/her thought processes and decision making [134, pp. 195-198]. This is a standard technique in usability studies. He/she was permitted to ask technical questions either to clarify how to carry out particular actions with the ADKT or to clarify the function of particular settings and custom variable types. However, all agent design choices were left to the participant alone. An audio recording was made of the session, and each time an agent was auditioned, the associated agent file and session file (i.e. that storing the learning configuration) were stored.

When the agent design was complete (as judged by the participant), a semi-
structured interview was carried out guided in part by the Cognitive Dimensions Questionnaire of Blackwell and Green\textsuperscript{3} [23]. The Cognitive Dimensions Questionnaire has previously been used in an interview context by its authors [23] and in the study of computer music software, also in an interactive context, by Duignan [73, 74].

9.2.2 Results

The Agent Design Process

To arrive at a satisfactory agent design, the participant performed eight design iterations. Each iteration involved arriving at a learning configuration, creating an agent and auditioning it. Here, we give an account of the design process, supported by quotations from the participant’s verbalisations along with records of the learning configurations used at each iteration.

Throughout the following, the learning configurations (see Tables 9.1-9.8) are represented in a similar tabular fashion to that used in the previous chapter. For custom variables, the labels, formulae (see Section 5.4.6, particularly Table 5.8 for an explanation of the notation) and VMM orders are shown, while for other (‘non-custom’) variables the formulae are replaced by the appropriate symbols. In each table, the variables are listed in priority order (see Section 5.4.5) with the highest-priority variable at the top, and the priorities given in superscripted parentheses beside the VMM orders. For each rule group, the the minimum support, $S_{\text{min}}$, the minimum confidence, $C_{\text{min}}$, and the maximum itemset size, $\#_{\text{max}}$, (see Section 5.4.4) are shown; along with a list of the variables included in the group.

The participant began by listening to the first of the four examples in its entirety. He/she then listened individually to each clip in the live set, rearranging the order of the tracks on-screen so that the five percussive instruments were together (hats, snare1, snare2, conga and kick) and the three melodic instruments were together (lead, bass, ceramic tiles). The participant did not listen to the remaining examples or

Chapter 9. A Study of the ADTK from the Perspective of End-User Programming

Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead</td>
<td>$p_1$</td>
<td>None</td>
</tr>
<tr>
<td>Bass</td>
<td>$p_2$</td>
<td>None</td>
</tr>
<tr>
<td>Ceramic Tiles</td>
<td>$p_3$</td>
<td>None</td>
</tr>
<tr>
<td>Snare1</td>
<td>$p_4$</td>
<td>None</td>
</tr>
<tr>
<td>Snare2</td>
<td>$p_5$</td>
<td>None</td>
</tr>
<tr>
<td>Hats</td>
<td>$p_6$</td>
<td>None</td>
</tr>
<tr>
<td>Conga</td>
<td>$p_7$</td>
<td>None</td>
</tr>
<tr>
<td>Kick</td>
<td>$p_8$</td>
<td>None</td>
</tr>
</tbody>
</table>

Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>${S_{\text{min}}, C_{\text{min}}, #_{\text{max}}}$</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Melodic</td>
<td>${0.01, 0.999, 4}$</td>
<td>${p_1, p_2, p_3}$</td>
</tr>
<tr>
<td>Percussive</td>
<td>${0.01, 0.999, 4}$</td>
<td>${p_4, \ldots, p_8}$</td>
</tr>
</tbody>
</table>

Table 9.1: The learning configuration for Agent01.

visually study the example data.

The first learning configuration (Table 9.1) comprised just two rule groups, one containing the five percussive instruments, and one containing the three melodic instruments. No temporal modelling was used. The participant listened to the resulting agent (AGENT01) for approximately 40 seconds, before returning to the Agent Designer. His/her primary complaint was the lack of ‘consistency’ in the ‘melodic ... and rhythmic variation’. Expanding on this, he/she mentioned the ‘the progression between the two different chords, it’d be nice to preserve some notion of that’ and noted that the melodic instruments were ‘out of synch’.

For the second learning configuration (Table 9.2), the lead and bass instruments were modelled using BLOCK custom variables with a block length of four (‘want to create more consistency ... might be able to use blocking to do that’). As described in Section 5.4.6, BLOCK custom variables can be used to model hypermetrical structure, to the extent that it is observed in the training data set. The only rule group used, included just the two BLOCK custom variables. Again, no temporal modelling was used. The participant listened to the resulting agent (AGENT02) for approximately
9.2. A Preliminary Usability Study of the ADTK

<table>
<thead>
<tr>
<th>Variables</th>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead x4</td>
<td>$p_9 = \lbrack \text{BLOCK}(4, p_1) \rbrack$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Bass x4</td>
<td>$p_{10} = \lbrack \text{BLOCK}(4, p_2) \rbrack$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Lead</td>
<td>$p_1$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Bass</td>
<td>$p_2$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Ceramic Tiles</td>
<td>$p_3$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Snare1</td>
<td>$p_4$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Snare2</td>
<td>$p_5$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Hats</td>
<td>$p_6$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Conga</td>
<td>$p_7$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Kick</td>
<td>$p_8$</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule Groups</th>
<th>Label</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>${S_{min}, C_{min}, #_{max}}$</td>
<td>${p_9, p_{10}}$</td>
</tr>
</tbody>
</table>

Table 9.2: The learning configuration for Agent02.

22 seconds, commenting that the lead and bass instruments ‘weren’t triggered very much at all.’

To arrive at the third learning configuration (Table 9.3), the only change made was to use 1st-order VMMs to model the two block variables (‘by adding a first order model, I hope to increase the events [more like] the examples’). For the audition of Agent03, the participant’s focus was on listening for ‘melodic components triggering ... more frequently ... and in synch’. He/she listened for approximately 50 seconds and stopped, commenting that the agent ‘did play the lead and basslines more often and ... it was ok,’ but, ‘not triggering enough ... I’d like it [melodic components] to be playing most of the time.’

While considering possible options for the fourth learning configuration, the participant made explicit his/her strategy of focussing on the melodic instruments first:

‘the rhythmic components aren’t necessarily as important to be consistent because ... they sound alright when they’re doing their thing, I don’t mind
the variation in the rhythmic components ... I think I would probably address the melodic structure, and then I would address the rhythmic components. That’d just be my ... personal strategy’.

He/she considered a variety of options including using a higher-order VMM (‘maybe I should try second order’); a COMBO custom variable of the bass and lead instruments; and a compound custom variable involving modelling such a COMBO with a BLOCK. However, the difficulty of predicting the results of these options was made clear:

‘What I’m kind of feeling at the moment is it’s just an exploration ... of what the combination of custom variables as well as grouping actually does, in a kind of a very ... yeah, I really just want to listen to the results of what’s happening ... it’s difficult to grasp every single bit in the process ... it’s quite a complex interaction that they’re having so it’s hard to, you know, mentally actually think about what I might specifically want it to do without just going through that iterative, kind of, process of just

<table>
<thead>
<tr>
<th>Variables</th>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead x4</td>
<td>$p_9 = \text{BLOCK}(4, p_1)$</td>
<td>1(1)</td>
<td></td>
</tr>
<tr>
<td>Bass x4</td>
<td>$p_{10} = \text{BLOCK}(4, p_2)$</td>
<td>1(1)</td>
<td></td>
</tr>
<tr>
<td>Lead</td>
<td>$p_1$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Bass</td>
<td>$p_2$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Ceramic Tiles</td>
<td>$p_3$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Snare1</td>
<td>$p_4$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Snare2</td>
<td>$p_5$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Hats</td>
<td>$p_6$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Conga</td>
<td>$p_7$</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Kick</td>
<td>$p_8$</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule Groups</th>
<th>Label</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>${s_{\text{min}}, c_{\text{min}}, #_{\text{max}}}$</td>
<td>${p_9, p_{10}}$</td>
</tr>
</tbody>
</table>

Table 9.3: The learning configuration for Agent03.
In this spirit, the participant defined the fourth learning configuration (Table 9.4) by creating a COMBO variable of the lead and bass instruments, modelled with a 2nd-order VMM. No other temporal modelling was used, and no rule groups were used. He/she listened for approximately 40 seconds, during which neither the bass or lead instruments were heard at all (‘still not achieving the desired result’).

For the fifth learning configuration (Table 9.5), the participant removed the COMBO custom variable and reinstated the two BLOCK custom variables to model the lead and bass instruments separately, but this time with 3rd-order VMMs. He/she listened to the resulting agent (AGENT05) for approximately 1’45” and commented

‘good! ... third order modelling on the blocks custom variables—certainly did seem to be—the progression was there ... now I do start to notice that the rhythmic components are probably varying too much ... yeah, no I liked that much better.’

Interestingly, it was only at this point that the participant seemed to become
explicitly aware that the learning configuration is not the sole determinant of an agent’s characteristics, but also the specifics of the training data set and the interactions between the two. He/she noted ‘the only thing I realised is that I should probably look at [the examples],’ and then opened one of the examples within the software to examine the variation in the bass instrument:

‘... I just wanted to look at how many zeros there were to how many ones, just to see what it’s actually doing ... you can see there that there’s large groupings of ones and large groupings of zeros, as opposed to four ones four zeros, four ones four zeros ... it [the bass] is really on for a lot [of a particular example] ... that’s interesting ... I hadn’t initially considered looking at that, but it seems to come to light.’

To create the sixth learning configuration (Table 9.6), the participant applied 3rd-order VMMs to all variables as an experiment (‘just to see what the results are ... I don’t have a mental picture’). He/she also re-introduced the ‘Melodic’ rule group comprising the three melodic instruments. This was the first instance in
which he/she dealt explicitly with variable priorities, giving the lead instrument block variable (Lead x4) the highest priority, and all other variables were given an equal, lower priority. The participant listened to the resulting agent (AGENT06) for approximately 2′22. He/she listened for specific characteristics, such as the re-introduction of the lead instrument after it had been removed (‘let me see if that comes in again’). Overall, he/she was satisfied with the the basic ‘consistency’ of the performance, particularly with respect to the lead and bass instruments:

‘that’s good ... that’s what I certainly wanted to hear from it ... 3rd order modelling seems to be somewhat more consistent but not necessarily in the rhythmic area ... it’s not making me think that third order is the absolute best for rhythmic modelling but it definitely, in terms of getting consistency for the lead and and bassline stuff it seems to be very good, it seems to be doing what I want it to do. I don’t necessarily notice a massive difference when I use third order as opposed to no modelling for the rhythmic stuff even though it is obviously more similar to the originals.’

Note that the participant did not appear to differentiate between the third order VMMs on the BLOCK variables (giving an effective history length of 12 bars: three groups of four), and a third order VMM on individual track variables (which has a history length of three bars). Related to this, he/she has a certain amount of faith in the modelling, trusting that the third order model resulted in performances more similar to the examples even though it was not obvious.

At this point the participant was asked to estimate his/her progress to achieving a satisfactory agent. Agreeing that the melodic and rhythmic components seemed to be internally ‘consistent’, he/she identified the next issue to be that of controlling overall dynamics:

‘using the summing custom variable ... that would be a really effective way of ... modelling dynamic ... response over time ... I mean it seemed
### Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead x4</td>
<td>( p_9 = \text{ BLOCK}(4, p_1) )</td>
<td>3(^{(1)})</td>
</tr>
<tr>
<td>Bass x4</td>
<td>( p_{10} = \text{ BLOCK}(4, p_2) )</td>
<td>3(^{(2)})</td>
</tr>
<tr>
<td>Lead</td>
<td>( p_1 )</td>
<td>3(^{(2)})</td>
</tr>
<tr>
<td>Bass</td>
<td>( p_2 )</td>
<td>3(^{(2)})</td>
</tr>
<tr>
<td>Ceramic Tiles</td>
<td>( p_3 )</td>
<td>3(^{(2)})</td>
</tr>
<tr>
<td>Snare1</td>
<td>( p_4 )</td>
<td>3(^{(2)})</td>
</tr>
<tr>
<td>Snare2</td>
<td>( p_5 )</td>
<td>3(^{(2)})</td>
</tr>
<tr>
<td>Hats</td>
<td>( p_6 )</td>
<td>3(^{(2)})</td>
</tr>
<tr>
<td>Conga</td>
<td>( p_7 )</td>
<td>3(^{(2)})</td>
</tr>
<tr>
<td>Kick</td>
<td>( p_8 )</td>
<td>3(^{(2)})</td>
</tr>
</tbody>
</table>

### Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>( { S_{\text{min}}, C_{\text{min}}, #_{\text{max}} } )</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Melodic</td>
<td>( {0.01, 0.999, 4} )</td>
<td>( {p_1, p_2, p_3} )</td>
</tr>
</tbody>
</table>

Table 9.6: The learning configuration for Agent06.

The participant then created the learning configuration shown in Table 9.7. This includes a \textit{SUM} custom variable representing the number of tracks playing. The lead and bass \textit{BLOCK} variables are still present, but having equal priority with the track variables \((p_1, \ldots, p_8)\) means that they may frequently be over-ridden by other variable selections. (Recall that when variables have equal priorities, their priority order is randomised at each decision point.) No rule groups were included in the learning configuration.

The participant prepared to audition \textit{AGENT07} with anticipation (‘I definitely think it’s getting more interesting, the further we go into this’). He/she listened to two agent performances, one lasting approximately 1’05”, and the other lasting...
approximately 1’20”. The first performance began with the *Kick* instrument playing on its own before other instruments were gradually introduced. The participant restarted the agent to see would a similar dynamic progression arise a second time (‘seeing whether the summing agent was actually really [causing the dynamic progression]’). Again the track began with a single instrument (*Hats*). The participant was clearly enthusiastic about the use of the *SUM* variable (‘I think the summing is really awesome, I really like having control in a non-direct sense of the dynamic content of the progression’). However, there were instances when the melodic instruments conflicted with one another due to the inappropriately set priorities reducing the efficacy of the *BLOCK* variables. In addition, the solo *Kick* instrument which began the first performance remained for an inappropriately long duration (16 bars) before other instruments were introduced.

The participant continued by addressing the problems due to the way in which the priorities were set:

‘I quite liked that ... may want to reintroduce the lead priority ... so that lead again takes priority, sum takes second, and everything else takes third ... bass would be the same as lead.’

This is reflected in the eighth learning configuration (see Table 9.8), which is identical to the seventh one, but for the altered priorities and the reintroduction of the *Percussive* rule group. The participant listened to *AGENT08* for approximately 1’20”, and commented

‘that’s really good, that’s even better again, I like that even more ... chord progression consistency ... dynamic range of events again was evident ... went from a simple to a more complex rhythm and then came back again slightly ... It’s interesting to reset the agent and listen again, and I feel I could almost do that a lot ... to kind of listen for consistency between takes ... consistency and variation ... observing that in fact my summing variable, my group and all this kind of stuff are actually kind
### Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tracks Playing</td>
<td>$p_{11} = \Sigma(p_{12}, \ldots, p_{19})$</td>
<td>$3^{(1)}$</td>
</tr>
<tr>
<td>Lead x4</td>
<td>$p_9 = \text{BLOCK}(4, p_1)$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Bass x4</td>
<td>$p_{10} = \text{BLOCK}(4, p_2)$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Lead</td>
<td>$p_1$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Bass</td>
<td>$p_2$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Ceramic Tiles</td>
<td>$p_3$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Snare1</td>
<td>$p_4$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Snare2</td>
<td>$p_5$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Hats</td>
<td>$p_6$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Conga</td>
<td>$p_7$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Kick</td>
<td>$p_8$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Lead Active</td>
<td>$p_{12} = \text{\textgreater ANY }(-1, p_1)$</td>
<td>None</td>
</tr>
<tr>
<td>Bass Active</td>
<td>$p_{13} = \text{\textgreater ANY }(-1, p_2)$</td>
<td>None</td>
</tr>
<tr>
<td>Ceramic Tiles Active</td>
<td>$p_{14} = \text{\textgreater ANY }(-1, p_3)$</td>
<td>None</td>
</tr>
<tr>
<td>Snare1 Active</td>
<td>$p_{15} = \text{\textgreater ANY }(-1, p_4)$</td>
<td>None</td>
</tr>
<tr>
<td>Snare2 Active</td>
<td>$p_{16} = \text{\textgreater ANY }(-1, p_5)$</td>
<td>None</td>
</tr>
<tr>
<td>Hats Active</td>
<td>$p_{17} = \text{\textgreater ANY }(-1, p_6)$</td>
<td>None</td>
</tr>
<tr>
<td>Conga Active</td>
<td>$p_{18} = \text{\textgreater ANY }(-1, p_7)$</td>
<td>None</td>
</tr>
<tr>
<td>Kick Active</td>
<td>$p_{19} = \text{\textgreater ANY }(-1, p_8)$</td>
<td>None</td>
</tr>
</tbody>
</table>

### Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>${S_{\min}, C_{\min}, #_{\max}}$</td>
</tr>
</tbody>
</table>

No rule groups.

Table 9.7: The learning configuration for Agent07.
of doing what I want them to do ... but I think I’ve kind of observed that iteratively.’

At this point, the participant considered the agent design complete (‘I think it achieves what I set out to achieve’). However, note that while this agent design does effectively capture both hypermetrical structure and dynamic structure, it has certain drawbacks. For example it is still possible for a performance to begin with a percussion instrument playing on its own for an extended period (as occurred when auditioning AGENT07). During the discussion which followed, the participant agreed that he/she had overlooked this issue.

**Participant Responses and Feedback**

Overall, the participant was enthusiastic about the ADTK. Comparing it to other software for creating generative music systems, he/she noted:

‘The rules remind me again of ... apps which already have a set of rules ... which aren’t malleable ... [the ADTK allows you to] define those rules yourself ... as opposed to someone’s pre-written rules for a generative process ... which I think is really nice ... as something that creates constant variation .. you can articulate yourself much more effectively with it ... with [ADTK] you feel like you’ve got a lot of control.’

Of the design process, he/she agreed that his/her strategy was to iteratively choose the most important problem with the agent’s performance and to address it:

‘taking ... the randomness that you have at the beginning ... and then gradually refining that back, chipping it down, you know, kind of sculpting it into something.’

The difficulties of clearly understanding the interactions between different custom variables were referred to on a number of occasions. The participant cited ‘combining multiple elements’ and ‘specific results of interactions’ as the kinds of
### Variables

<table>
<thead>
<tr>
<th>Label</th>
<th>Formula / Symbol</th>
<th>VMM Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead x4</td>
<td>$p_9 = \text{BLOCK}(4, p_1)$</td>
<td>$3^{(1)}$</td>
</tr>
<tr>
<td>Bass x4</td>
<td>$p_{10} = \text{BLOCK}(4, p_2)$</td>
<td>$3^{(1)}$</td>
</tr>
<tr>
<td>Number of Tracks Playing</td>
<td>$p_{11} = \Sigma(p_{12}, \ldots, p_{19})$</td>
<td>$3^{(2)}$</td>
</tr>
<tr>
<td>Lead</td>
<td>$p_1$</td>
<td>$3^{(3)}$</td>
</tr>
<tr>
<td>Bass</td>
<td>$p_2$</td>
<td>$3^{(3)}$</td>
</tr>
<tr>
<td>Ceramic Tiles</td>
<td>$p_3$</td>
<td>$3^{(3)}$</td>
</tr>
<tr>
<td>Snare1</td>
<td>$p_4$</td>
<td>$3^{(3)}$</td>
</tr>
<tr>
<td>Snare2</td>
<td>$p_5$</td>
<td>$3^{(3)}$</td>
</tr>
<tr>
<td>Hats</td>
<td>$p_6$</td>
<td>$3^{(3)}$</td>
</tr>
<tr>
<td>Conga</td>
<td>$p_7$</td>
<td>$3^{(3)}$</td>
</tr>
<tr>
<td>Kick</td>
<td>$p_8$</td>
<td>$3^{(3)}$</td>
</tr>
<tr>
<td>Lead Active</td>
<td>$p_{12} = &gt;\text{ANY}(-1, p_1)$</td>
<td>None</td>
</tr>
<tr>
<td>Bass Active</td>
<td>$p_{13} = &gt;\text{ANY}(-1, p_2)$</td>
<td>None</td>
</tr>
<tr>
<td>Ceramic Tiles Active</td>
<td>$p_{14} = &gt;\text{ANY}(-1, p_3)$</td>
<td>None</td>
</tr>
<tr>
<td>Snare1 Active</td>
<td>$p_{15} = &gt;\text{ANY}(-1, p_4)$</td>
<td>None</td>
</tr>
<tr>
<td>Snare2 Active</td>
<td>$p_{16} = &gt;\text{ANY}(-1, p_5)$</td>
<td>None</td>
</tr>
<tr>
<td>Hats Active</td>
<td>$p_{17} = &gt;\text{ANY}(-1, p_6)$</td>
<td>None</td>
</tr>
<tr>
<td>Conga Active</td>
<td>$p_{18} = &gt;\text{ANY}(-1, p_7)$</td>
<td>None</td>
</tr>
<tr>
<td>Kick Active</td>
<td>$p_{19} = &gt;\text{ANY}(-1, p_8)$</td>
<td>None</td>
</tr>
</tbody>
</table>

### Rule Groups

<table>
<thead>
<tr>
<th>Label</th>
<th>${S_{\min}, C_{\min}, #_{\max}}$</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percussive</td>
<td>${0.01, 0.999, 4}$</td>
<td>${p_4, \ldots, p_8}$</td>
</tr>
</tbody>
</table>

Table 9.8: The learning configuration for Agent08.
things that require the most mental effort. He/she mentioned ‘unwanted interactions’ as being the principle hidden dependencies.

The other issue that repeatedly arose was that of gaining a good understanding of the agent’s behaviour (‘a full understanding is hard to see’). The participant noted that

‘If you listen to it once, I feel like you get a good picture of maybe what it’s doing but if you listen to it twice, you get a much better view, and then the third time, even better again ... so you don’t get stuck in the details but see that it’s broadly doing the right thing.’

He/she suggested visualising the output as in the session view of Ableton Live ‘to see patterns of variations.’

Relating to general usability issues, the participant cited ‘not being able to state a rule’ as being a strange feature of the software. In addition, he/she suggested that the notation of custom variables in particular should be ‘more musically relevant’, adding ‘once explained, it is easier, but their relevance is clearer when actually listening to results.’

### 9.3 Discussion

#### 9.3.1 Attentional Investment

The CDs analysis highlighted that the ADTK is extremely abstraction-hungry, requiring the mastery of a number of abstractions, including Markov models and custom variables, before it can be used successfully. Blackwell [19] notes that this property ‘tends to constitute an initial obstacle for end-users, because it demands a higher level of attentional investment.’ He argues that a higher level of attentional investment is an obstacle because of the risk of not achieving one’s goal; a user does not want to make the (attentional) investment if there is a significant risk that it will fail to pay off. To minimise this risk, Blackwell suggests that the user should be able to, insofar as is possible,
• predict the inference that will result from examples;

• see the inferred program;

• anticipate the effects of inferred program; and

• modify the inferred program.

In the following, we draw together the CDs analysis and the results of the usability study with reference to these requirements. We divide the discussion into two parts according to the two phases of an agent design iteration: creation of the learning configuration and evaluation of the agent. Note that in the language of EUP, the first of these relates to the issues of notation, abstraction and debugging, while the second relates to representation of the inferred program (see the chapter introduction).

#### 9.3.2 Creation of the Learning Configuration

The CDs analysis presented in Section 9.1 highlighted four particular aspects of the ADTK notation relating to creating a learning configuration. These were (i) the frequent presence of hidden dependencies, (ii) the emphasis on abstraction, (iii) the requirement for hard mental operations due to the first two, and finally (iv) the lack of role-expressiveness and closeness of mapping between the notation and the target domain. The participant specifically mentioned the first, third and fourth of these specifically in his/her responses and examples of the first two can be found in the account of the agent design process.

A number of the problems encountered by the participant in the usability study were due to hidden dependencies. For example, the disruption of the hypermetrical structure observed using AGENT07 was due to unforeseen conflicts between the SUM variable controlling the number of tracks playing, and the BLOCK variables modelling the four-bar structure. There were also hidden dependencies between the learning configuration and the training examples. When the COMBO variable used in AGENT04 did not have the desired effect, this was because the VMM order was
not high enough to capture the patterns in the data. Though the participant found an alternative strategy that of using the COMBO variable might have been effective with a higher VMM order. The participant, therefore, could not effectively modify (or debug) the inferred program, but instead had to revert to a previous learning configuration (AGENT03) and develop from there (AGENT05).

It was possible to see the participant becoming aware of these dependencies over the course of the study. He/she first inspected the training examples approximately half-way through. In addition, he/she noted the ‘unwanted interactions’ in answering questions after the design process. That the participant would gain this awareness relatively quickly is promising but it is unlikely that this would be true of users in general, particularly with more sophisticated agent behaviours. Even with this awareness, the task was still difficult; the participant made numerous references to empirical experimentation and not having a ‘clear mental picture’ of the results of particular actions (inability to predict the inference).

During the agent design process, the participant was able to make effective use of many of the abstractions present in the ADTK. For example, he/she arrived at a design that took advantage of the SUM and BLOCK custom variables to model musical dynamics and hypermetrical structure, respectively. It is interesting to note that he/she achieved this without having a full understanding of the subtleties in every case, but rather with broad expectations of the effects of different actions. For example, while he/she drew a clear connection between Markov model order and the extent to which the agent would mimic the training data, he/she did not study the data to try to optimise the order, but rather arrived at the order by trial and error. Furthermore, he/she did not differentiate between the VMM orders of BLOCK variables and those of other variables, despite there being a significant distinction. Thus, while a detailed understanding may be required for certain debugging tasks, a more high-level understanding might be sufficient in many situations.
9.3.3 Evaluating the Agent

Both the results of the previous chapter and the CDs analysis presented here identified issues with the evaluation of agents. In all but the simplest of cases, it is not possible to fully predict the model that will be inferred. The interactions between the different model components—training examples, custom variables, VMMs and rule groups—are in general too complex to fully grasp, thus empirical testing of an agent design will always be required.

In the ADTK, the discovered rules are printed to the screen for inspection when an agent is trained, but this is the extent to which an agent model is ‘visible’ to the user. In the usability study reported here, the participant did not attempt to gain any insights from the printed rules. Essentially, for him/her it was not possible to see the inferred program. Instead, the participant relied solely on auditioning to gain understanding of the agents. It is noteworthy that the participant’s listening durations tended to increase as the agent improved and the learning configuration became more sophisticated; it took very little time to find major flaws at the beginning, but then it started to take longer. In addition, when a good performance was encountered, repeated listening was required to ensure that it did not arise by chance. Thus, while it is relatively easy to find flaws, it can be extremely difficult to empirically show that an agent is perfect.

The participant identified these issues in his/her feedback, and suggested generating numerous performances offline and displaying them graphically to the user. This is a feasible idea and will be included in future work. As well as increasing visibility, it could be expanded to allow the juxtaposing of different agents for comparison.

9.4 Conclusion

In this chapter, we have presented an analysis of the ADTK from the perspective of end-user programming, and reported on a preliminary usability study. With
reference to research question III-(v) (see Section 1.6.2), we identified a variety of components of the ADTK notation that demand further attention. In particular, the hard mental operations arising both from the level of abstraction present in the ADTK and the hidden dependencies that frequently arise, must be addressed. This might be done by introducing heuristics into the software to help prevent unwanted hidden dependencies from arising and also to make these more apparent when they become important. In addition, it would clearly help to improve the role-expressiveness and closeness of mapping of the notation, particularly custom variables. As well as issues with the notation, we highlighted difficulties in evaluating agents both in the CDs analysis and the usability study. These results lend support to the findings presented in the previous chapter.
Chapter 10

Discussion

10.1 The ADTK as a Creativity Support Tool .......................... 354
  10.1.1 Design Principles ................................................. 355
  10.1.2 Generating New Musical Ideas ................................. 361

10.2 The ADTK in the Context of Interactive Machine Learning .... 362
  10.2.1 Effectively Illustrating the Learned Model ......................... 362

10.3 Summary of Future Work ............................................. 365
  10.3.1 Additional Components for the ADTK .......................... 365
  10.3.2 HCI Research .......................................................... 368

10.4 Conclusion .............................................................. 369
In this chapter, we discuss the Agent Designer Toolkit first through as a creativity support tool (Section 10.1) and with respect to the interactive machine learning paradigm (Section 10.2). We use the issues arising in each of these areas as a framework for drawing together the results of the previous three chapters, and identifying questions to answer in future research. We then conclude with a summary of our proposed future work (Section 10.3).

10.1 The ADTK as a Creativity Support Tool

Creativity support tools are ‘tools that enable people to express themselves creatively and to develop as creative thinkers’ [147]. Such tools may take many forms (e.g. a pencil and paper for sketching) but particularly relevant to discussion of the ADTK are software creativity support tools, specifically, ‘composition tools’ which are those that people can use to ‘generate, modify, interact and play with’ [147] particular (digital or physical) artefacts or types of artefact. In the case of the ADTK, musical agents are the artefacts in question. However we view the ADTK not in isolation, but as an important new part of a rich modular toolset for creating interactive and generative systems, that also includes Max and Ableton Live. These may be considered collectively as a set of creativity support tools for ‘composing’ interactive and generative music systems. In particular, the following two subsets of tools were used for developing such systems in the three preceding chapters:

1. ADTK + Max (+ anything that can be communicated with via OSC)

2. ADTK + Max for Live + Ableton Live

For succinctness, we refer to the first of these as ADTK+Max and the second as ADTK+Live. In the following, we will emphasise ADTK+Live, since it is most relevant to non-programming end users.
10.1.1 Design Principles

It is challenging to rigorously evaluate creativity support tools (see, e.g. [160, pp. 10-24], [44]) and the scope of the studies reported in the previous three chapters do not permit an exhaustive characterisation of the ADTK in this regard. However, we note that Resnick et al [147] do ‘consider diversity of outcomes as an indicator of success’ and the studies presented in this thesis have illustrated the potential of ADTK+Live and ADTK+Max (particularly the former) with respect to diversity in musical style and context.

In addition, Resnick et al [147] outline a set of design principles for developing creativity support tools. Following Fiebrink’s discussion of the Wekinator (computer music software with which the ADTK has aspects in common; see Section 5.6.4) in [80, Ch. 9], we use a selection of these principles to frame the following discussion.

**Support Exploration**

*This design principle refers to the need for a creativity support tool to make it easy for users to try different ideas and backtrack when necessary, and additionally that the features are clearly presented (‘self revealing’) and that the software as a whole is ‘pleasurable and fun to use’* [147].

The case studies presented in Chapter 8 highlighted the speed with which agents could be created. Four of the five agents—the Backgammon agent was an exception—were developed (including a number of design iterations and auditions of intermediate designs in some cases) in about an hour or less. Furthermore, initial designs were usually available to run within minutes of the examples having been recorded. This indicates that once a set of feature extractors and generators has been created (often as simple as creating a few clips in Ableton Live) a ‘rough-and-ready’ generative or interactive system can be created almost immediately.

This contrasts starkly with previously existing tools for designing such systems. For example, in general a Max user is required to program the decision making logic by hand. Moreover, if the agent does not perform as expected, extensive re-
programming might be needed to improve it. Using the Agent Designer, changes are very quick to make, so while we do envisage scenarios in which the required changes will not be obvious to a user, it will generally be the case that the steps, when identified, will take very little time to implement and then audition.

To some extent, the ADTK does place constraints on the design workflow. For example, design is most straightforward when the feature extractor and generator components of the system are finalised before the agent design process begins. This is because recorded examples may cease to be meaningful if significant changes are made to the feature extractor or generator. However, it is straightforward to delete old examples and add new ones while maintaining the learning configuration, so these constraints are not entirely rigid.

Also related to the support of exploration is the basic usability of the interface. At the time of writing, the ADTK interface lacks certain standard features such as keyboard shortcuts, and importantly, undo functionality, that are considered to be central to supporting exploratory search. These will be added in due course.

More important, however, are the usability issues identified in the previous chapter which stem from the requirement to design the learning configuration itself, rather than from the specifics of the prototype user interface. The preliminary user study (see Section 9.2) supported our position that these issues do not make the software unusable, however, there is clearly room for significant improvement. We have made a number of proposals with regard to the provision of a more innovative user interface aimed at improving the usability of the ADTK and they will be systematically explored in future research.

Finally, in our experience of using the ADTK, the learning process is very fast (rarely taking more than 10 seconds to complete). However, as mentioned with regard to the design of the Backgammon agent (see Section 8.2.5), it can occasionally take a long time to perform the SAT \(\rightarrow\) BDD conversion that is required before an agent can be used. In fact, as reported in Section 8.3.4, this impeded the free exploration of alternative designs because of the desire to avoid problematic ones. As
noted there, a straightforward improvement would be the provision of a progress bar to give the user an indication of how long this process will take, and a ‘cancel’ button if it is progressing too slowly. However, methods for speeding up this conversion (or making it unnecessary) might constitute an additional avenue for future work.

**Design with Low Thresholds, High Ceilings and Wide Walls**

This design principle refers to the need for creativity support tools that are easy to begin using (low threshold); capable of supporting sophisticated designs (high ceilings); and support and suggest a wide range of explorations (wide walls).

We plan to develop the ADTK further, in particular, to add a higher-level user interface. While we have reported substantial findings that will inform this work, only a tentative step has been taken so far: a button is available that automatically applies a particular preset learning configuration (this button is labelled ‘Learning Configurations’ and can be seen in Figure A.4). Nevertheless, even without a higher-level interface, we speculate that there is a relatively low threshold to designing a simple agent with the ADTK+Live system, since the Ableton Live interface is familiar to many musicians and once some examples have been recorded, a simple agent can be designed very quickly by following the steps to use that preset learning configuration or set up a simple one from scratch. Of course, should such a learning configuration be unsatisfactory, the process abruptly becomes more complex. This can be related to the fact that the ADTK is abstraction hungry and has a high abstraction barrier (see Section 9.1).

In the previous chapters the ‘high ceilings’ of the ADTK were demonstrated through the development of a variety of sophisticated interactive and generative systems. While some of this sophistication can be attributed to the affordances of Ableton Live, the ADTK provided the essential means of designing the arrangement-level decision making components. We do not claim that the agents were as sophisticated as many of the hand-coded ones mentioned in Section 1.3. However, the ease and speed with which ADTK agents can be developed, indicates that ADTK+Live
makes the development of interactive and generative system available to a much wider range of users: we consider it almost a certainty that to non-programming end users, the ‘ceilings’ are much higher than with any other set of tools.

Finally, the work reported in Chapters 7 and 8 showed that it is possible to use the ADTK+Live set of tools to create a substantial variety (wide walls) of interactive and generative systems, at least in the context of contemporary electronic music production. In Chapter 7, these included ones relevant to various mainstream musical genres (‘techno’, ‘electronica’, ‘ambient’) as well as contemporary art music (‘electroacoustic’), and in Chapter 8, systems were created for a variety of different performance contexts (generative systems, player-paradigm systems for improvisatory contexts, live algorithms).

**Make it as Simple as Possible and Choose Black Boxes Carefully**

This is an amalgamation of two design principles that are entirely interdependent in the case of the ADTK. The first refers to making the software easy to use, while the second refers to the selection of ‘black boxes’, i.e., software features that are opaque to the user.

An open question with regard to the ADTK is how intuitive the user interface can be made without compromising the modelling capabilities of the software. However, the layered interface design proposed in Section 7.4.2 would allow considerable flexibility in this regard. We have arrived at an initial set of preset learning configurations (black boxes) for the top layer (i.e. that designed for least experienced users) and made a methodological recommendation involving repetitions of the study described in Chapter 7 to refine these and add more (see Section 7.4.2).

We envisage that the lowest layer of a layered interface (that for the most experienced users) would be reasonably similar to the interface currently implemented (see Appendix A). However, a number of obvious improvements could be made, such as the use of terminology more relevant to music (e.g. BLOCK custom variables might become METRICAL STRUCTURE variables). In addition, the provision of documentation, including suggestions for achieving certain decision making behaviours.
would doubtless be of benefit.

Perhaps the most interesting future work relates to the design of an intermediate layer, which we envisage would allow behavioural ‘traits’ to be selected individually (again, see Section 7.4.2). To this end, we have identified a number of common modelling sub-problems and techniques for solving them, such as capturing dynamic structure, simultaneous changes between different control variables and establishing sets of values that are musically equivalent (see Section 7.3). Our view is that these could underly a distinct behavioural trait (e.g. ‘these instruments should adhere to a four-bar hypermetrical structure’; ‘these clips should be treated as musically equivalent’, etc.). Questions remain, however, as to how to solve usability issues related to hidden dependencies (see Section 9.1) between different traits.

**Support Many Paths and Many Styles**

This design principle refers to the support of different approaches to achieving a given goal, including different cognitive tendencies (‘logical, analytical’ in contrast to ‘holistic, intuitive’) [147].

Much of the discussion of the agent design process in Section 8.3.3 related to the difficulties associated with analysing or introspecting about example performances in order to arrive at stylistically salient patterns that could be precisely defined. This type of thinking would seem incompatible with the ‘soft approach’ taken by ‘holistic, intuitive’ thinkers1.

One of the promises of the programming by example paradigm is that it would mitigate the need for analytic thinking about the largely intuitive task of performing music. However, due to the extremely small training data sets from which the Agent Designer is required to learn, this promise is not entirely fulfilled: the user is required to perform the important feature selection step and this requires the bridging of a

---

1It is also conceivable that the Likert-type questions in the study reported in Chapter 7 required a type of analytical approach that was unnatural to some participants (e.g. differentiate between ‘vertical’ and ‘short-term’ style similarity), and that this may have contributed to the discrepancies in some cases between the responses to the Likert-type questions and those to the free-form answers.
gap between musical ideas and precise technical specifications. The music ideas must certainly come from the user, however, there is considerable scope for the software to aid in the translation to technical specifications.

This translation from musical ideas to technical specifications (i.e. to learning configurations) is the purpose of our proposed higher-level interface. A very rough translation might be performed by the selection of presets (appropriately named), however, the aforementioned intermediate layer of the interface is critical in this regard. Further research is required, which would focus on establishing an effective and concrete mapping (translation) between musicians’ descriptions of their example performances (perhaps their responses to a series of questions) and the modelling constructs established in Chapter 7 to capture specific musical patterns.

**Support Collaboration and Support Open Interchange**

*This is an amalgamation of two design principles referring to (i) support for different participants to collaborate on the same project and also for sharing of projects and techniques, and (ii) support for interoperation with other relevant tools.*

One tantalising aspect of the ADTK is the potential for sharing (i) agents, (ii) learning configurations and (iii) training data sets. While agents are generally tailored to specific sets of material, it would be straightforward to add components that would make it possible to map the variables output by an agent, to the control of other parameters. Another possibility would be to take an approach similar to that used to create the *Backgammon* agent, whereby an agent was designed for a template Live set, which was then populated with material. The idea of sharing agents relates directly to the that of a *behavioural object* proposed by Bown et al [32], which is a transferable and shareable software entity embodying a particular dynamical behaviour.

The ADTK also makes it possible to save Agent Designer *session* files which contain the learning configuration along with the training data set. Though we envisage a scenario in which users will not be *required* to engage directly with learning configurations, this facility is important for collaborative work. In addition,
for users that do engage with low-level learning configurations, transferring ideas is clearly straightforward; the representation of learning configurations adopted in the previous three chapters (see, e.g. Table 8.4) shows the economy with which learning configurations can be conveyed. Finally, the software supports exporting training examples to, and importing examples from, comma-separated-value files which can be prepared or manipulated in any text editor or spreadsheet software.

### 10.1.2 Generating New Musical Ideas

In addition to examining the ADTK as a creativity support tool for designing interactive and generative music systems, it can be seen as a tool for generating new musical ideas for music composition, that is, outside of a performance context. One important outcome of the study presented in Chapter 7 was the strong potential for ADTK agents to generate new musical ideas and the value placed on this by the participants. For each agent type used in the study, participants’ median level of agreement was moderate or strong, that an agent could give them new ideas for performing with their musical material. In this regard, it can be related to functionality available in many digital audio workstation software packages, such as the *Logical Editor* in Cubase [164], as well as tools for computer-aided composition that can generate variations on a given musical idea, such as the *Variator* [158]. However, ADTK agents, uniquely, perform arrangement-level musical decision making and as noted in Chapter 7, this can give rise to different kinds of musical ideas. For example, one participant noted that an agent found interesting ways of ‘mingling … filtered and non-filtered drum parts … that goes well with the vocal’; this is an ‘arrangement-level’ idea. While the behaviour of the RANDOM agent (used in the study) in particular can be implemented directly in Ableton Live, those of the other types of agents cannot and there were indications that the types of new ideas were related to the types of agents used (see Section 7.2). The use of the ADTK as a generator of musical ideas in a compositional context is not one of the focusses of the research reported here, however, it could provide an avenue for future work.
### 10.2 The ADTK in the Context of Interactive Machine Learning

We previously described the ADTK as a hybrid between standard machine learning and interactive machine learning (IML; see Section 5.6.4). Amershi et al identify three important design challenges with respect to interactive machine learning systems [6], and they are relevant to the ADTK:

1. How to effectively illustrate of the current version of the learned model.
2. How to guide end users to take appropriate actions to arrive at a better model.
3. How to enable efficient and lightweight end user exploration of multiple potential models.

The second and third of these have been discussed already. The challenge of guiding end users to improving the model is central to our future study of the ADTK, and has been discussed under various headings in the previous section. Enabling the efficient and lightweight exploration of multiple models largely concerns many of the usability issues relevant to the ‘Support Exploration’ design principle, also discussed in the previous section. The first challenge listed above is strongly related to the problem of representation in end-user programming (see the previous chapter). We can use the autoethnographic study of Chapter 8 to make informed proposals with respect to effective representations, or illustrations of the learned model.

#### 10.2.1 Effectively Illustrating the Learned Model

Using the ADTK, the primary means by which a user can gain an understanding of the current model of musical performance is by auditioning it. This has the advantage of being completely transparent, meaning that the user is not required to perform any translation between the representation and the sound that might be output by the system, as would be required using, for example, a visual representation of the system’s output. However, it has the disadvantage of being real-time and therefore it
is time consuming (and, we found, fatiguing) to gain an understanding of the space of possible performances.

In the case studies described in Chapter 8, we highlighted two examples of our having an incomplete understanding of the model embodied by an agent. In other words, we did not have a good enough idea of the space of possible things that the agent could do. The first was discovered in the performance with the Automato system, when the Poc Tchk generator was used more than expected. The second was with the Isidores system, when the interaction between the Control Routing Config and Sequencer 1 Mode variables prevented the system from producing any sound for a period during the performance.

In both of these cases, the system had been auditioned, however, the difficulty of auditioning a probabilistic system is that it may by chance produce performances that are ‘statistical outliers’. This is problematic whether the outlier is particularly good (i.e. meeting or exceeding our expectations) or bad (failing to do so), since in the former case it may lead to the use of a flawed agent in a performance and in the latter, it may lead to unnecessary efforts to fix an agent that is extremely unlikely to perform unsatisfactorily. To gain a sufficient understanding of a model of performance may in some cases take an extremely long time and this is even more problematic in the case of an interactive system, when the musician with whom the system interacts must play with it for the duration of the audition (though recordings of the musician may provide a reasonable substitute or starting point).

While we do not envisage removing audition as a means to understand the model embodied by an agent, there is a clear need to supplement it with other information. The ADTK already presents part of the learnt model to the user by printing to the screen the set of association rules that has been learnt. When the set of rules was relatively small (perhaps around thirty or less) we found in most cases, that it was useful to inspect it in order to better understand the learnt model. In the following, we propose a number of other features which could be added to the software to aid user understanding of the learnt model.
First, as was suggested by the participant in the usability study of the previous chapter, the software could generate a number of performances offline and display them to the user in a way that is similar to our method for displaying training data sets in this thesis (see, e.g. Figure 8.6). This grid-like display of performance data, with variables (often instruments, or tracks) on the vertical axis, and time on the horizontal axis, is similar to the way in which digital audio workstation (DAW) software displays musical arrangements. For this reason, we envisage that it would be readily understood by users familiar with DAWs\(^2\).

With the aim of giving the user a more comprehensive understanding of the space of possible performances, this method might be enhanced byopaquely generating a large number of performances and displaying only a small selection of those that are most diverse according to some metric. This can be related to the \textit{split} method described in [81] for illustrating the model learned by a supervised machine learning algorithm to distinguish between two classes of images. Using this method the user is shown only two small sets of images to illustrate the model. One set contains images to which the model attributes a high certainty of being in one image class, and the other set contains images to which the model attributes a high certainty of being in the other image class. This method was shown to be more effective for illustrating the learnt model, than an alternative method in which the users were shown the performance of the model on the entire training data set. One explanation posited for this is that with the \textit{split} method, users spent more time fixing the most important errors being made by the model (i.e. when it shows a high certainty that an image is in the incorrect class) rather than trying to fix relatively minor problems with the model (improving its performance with respect to genuinely ambiguous images, for example). This might also apply in the ADTK: if the user were shown a diverse range of possible performances, he/she may improve the model most efficiently by improving it with respect to the most frequently arising problems.

\(^2\)When the ADTK is used inside Ableton Live, the performance data could be displayed using the same colours used to represent the clips in the Ableton Live set (these colours can be retrieved using the Max for Live API).
10.3 Summary of Future Work

10.3.1 Additional Components for the ADTK

We have proposed a number of improvements and additions to the ADTK. Among them are the features for visualising generated performances, and features for easily remapping an agent’s output to be used in new contexts (both were discussed

Indeed, even without such information, the participant of the usability study in the previous chapter took a very systematic approach to designing an agent, addressing the most obvious issues one by one.

An additional way in which the system could illustrate the learnt model to the user would be to display simple statistics about output performances. For example, after opaquely generating a set of performance data, the system could display histograms of the frequencies with which different variable values arise in the training data set and also in the generated data. The user would then decide whether the discrepancies between the two require attention. For example, if this had been done for the Automato system, the tendency of the system to overuse the Pvoctchk generator might have been identified (see Figure 10.1).

Figure 10.1: Showing simple statistics to improve a user’s understanding of the model. (a) a histogram showing the frequencies with which the Pvoctchk variable has the values 0 and 1 in the training data set. (b) a histogram showing these frequencies in performances generated using the agent. The value 1 appears approximately twice as often in the generated performances.

10.3 Summary of Future Work
earlier in this chapter). In the following, we list other proposals made the preceding chapters and suggest how they might be implemented.

**Macro Time Scale Organisation**

In Section 7.4.1, we proposed additional capabilities to model musical sections, in other words, musical structure on the *macro* time scale (see Section 1.2). A straightforward implementation of this would simply allow the user to select and label separate sections in each training example. A separate model would be then trained for each section.

To use these models for performance, two issues would need to be addressed. The first is that there is continuity between sections, that is, when one model is exchanged for another, that an abrupt change is not imposed where it is not stylistically appropriate. This could be achieved by maintaining the VMM histories across section changes, so that continuity is maintained, at least to the extent that it is in the training examples.

The second issue is that of choosing when to switch between models (or sections), that is, how to perform *macro*-level musical decision making. There are a number of possibilities with regard to this. One is that a musician manually chooses when one section ends and another begins. This might be appropriate where an agent is used by a musician during live performance. Another possibility is to simply copy the durations of each section from one of the training examples. A third possibility, mentioned previously is that a low-order Markov model or formal grammar is used on a longer time scale to switch between models, the former having been used, for example in the GEDMAS system for generating electronic dance music [10], and the GRI system of Morales et al [130]. Finally, for interactive contexts, it may be worthwhile to explore the use of partially observable Markov decision processes (see Chapter 3) to model sectional changes.
10.3. Summary of Future Work

**Durational Bounds and Breaks in Hypermetrical Structure**

Two other suggestions were made in Section 7.4.1 for improving the modelling capabilities of ADTK agents. To place durational bounds on the length for which a parameter might go unchanged would be straightforward, requiring an additional \texttt{NOT EQUALS} constraint to be added when a bound might be exceeded. To allow a more flexible modelling of hypermetrical structure, as suggested previously, the decision point count, that determines the alignment of \texttt{BLOCK} custom variables could be modified so that it is only constrained to a particular value when certain hypermetrically important instruments (or patterns) are playing. This would allow, for example, introductory sections and breakdowns of odd lengths.

**Gesture Quantizer**

For two participants in the study reported in Chapter 7, it was required to post-process the example performances so that the agent could be trained to manipulate gestural parameters (see Section 7.1.2). A parameter was considered to be gestural if

1. its value was modulated between decision points, possibly continuously, and/or
2. it was continuous-valued, and its values could not be rounded to integers.

We propose to augment the Agent Designer with a gesture quantizer for recording such parameter data and automatically creating a dictionary of gestures for each gestural parameter. In other words, we propose to automate the process that was done manually for the study in Chapter 7. Gestures in the training data could then be replaced by integers indexing into the dictionaries and these integers would be suitable for inclusion in ARL analysis and they could be modelled with VMMs. A starting point would be to use vector quantization techniques (see, e.g. [88]) to derive prototypical gestures for gestural parameters of the first type, and \textit{k}-means clustering or Gaussian mixture models (see, e.g. [92, Ch. 14]) to derive a set of typical values for those of the second type.
**Gesture Synthesizer**

A *gesture synthesizer* would be required to convert gesture dictionary indices into parameter control gestures during performance. A basic implementation would simply play back the gestures stored in the dictionary (this is what was done for the study in Chapter 7). However, if Gaussian mixture models were used, for example, random fluctuations could be added to give subtle variation to the output.

Other useful features in a gesture synthesizer might enable the user to precisely manage the timing of parameter changes. For certain parameters, it is preferable to change the value just before the beat, rather than precisely on it. For example, if the introduction of a ‘bass-cut’ filter (commonly used in techno music [41, p. 54]) coincides exactly with a beat, unwanted artefacts can be introduced. It is preferable to introduce it (and remove it) just before a beat. It would a straightforward to allow a user to select which parameters are of this type.

A final feature related to the gesture synthesizer might be the facility to randomise the timing of parameter updates. This was implemented for the participant using Max in the study of Chapter 7, in order that parameter changes not be so regular as to impose an unwanted pulse on the music generated by the agent. It would also be straightforward to allow the user to specify a range of values from which random delays could be chosen to apply to parameter updates.

**10.3.2 HCI Research**

Along with improvements to the ADTK itself, a number of proposals have been made for further HCI studies of the software. In particular studies will be carried out to research:

- The design and selection of a set of preset learning configurations to provide starting points (and perhaps solutions) for a wide variety of musical contexts.

- The identification of additional common modelling sub-problems and techniques for solving them using the ADTK (see Section 7.3).
10.4 Conclusion

- The construction of a ‘mapping’ or translation from descriptions attributed to example performances to modelling techniques.

- The adoption of a suitable set of musically-informed terminology for use in the Agent Designer interface.

Such aims might be best fulfilled by a participatory design study (see, e.g. [109, Ch. 9]) or further case studies in which the author did not take an active role.

10.4 Conclusion

In parallel with the research outlined above, we plan simply to make the ADTK publicly available along with a set of instructional videos and demonstration compositions in Ableton Live. This would inevitably lead to further insights about its use and how it could be improved, and help elicit feedback on its implementation so far.
Chapter 11

Conclusion

The aims of this work were to develop general purpose methods to support the design of arrangement-level musical decision making by non-programming end users. We identified the arrangement level of decision making as particularly important because it is a central process, not only in interactive and generative systems developed in the context of experimental music, but also in the performance of contemporary electronic music. Moreover, we highlighted the wide range of representations of musical information (knowledge representations) that arrangement-level decision makers can be required to accommodate, and thus, that general purpose methods would be required to support the design of such decision makers.

11.1 Summary of Findings

Below is a summary of the main findings of this thesis. We begin by taking in turn each of the research questions listed in Section 1.6.2.
I. To investigate the affordances of the partially observable Markov decision process (POMDP) as a framework for designing musical agents with respect to the requirements listed in Section 1.6.1

We began by studying the partially observable Markov decision process (POMDP) as a framework for musical decision making. The question we asked was whether a POMDP could be used to parametrically design arrangement-level musical decision making, that is, whether useful agents can be designed by adjusting the numerical parameters of the model. We showed that the POMDP can be used to design the responses of an agent to particular musical input. For example, we demonstrated that, simply by adjusting the reward function of the model, an agent’s behaviour could be varied between two extremes that could be characterised as ‘cautious’ and ‘risk-taking’, respectively, when faced with uncertainty about the musical key in which it should play. However, we were unable to extend our exploration of POMDPs to more general arrangement-level musical decision making. It remains unclear what representations of musical information should be used, and in addition, if appropriate representations were found, how to design the reward function.

II. To design, implement and evaluate a general-purpose, history-based system for modelling arrangement-level musical performance, based on the similarity measure for sequential patterns

We then turned from parametric design to the alternative paradigm of programming by example. We developed the similarity-based musical agent, which uses a novel instance-based machine learning algorithm (based on the similarity measure for sequential patterns) for performing arrangement-level musical decision making based on a set of example performances. A preliminary evaluation of the method showed that it is capable of learning at least simple arrangement-level musical behaviours, but that too much training data is required. While it is possible that
other techniques for programming by example would be more successful, we turned instead to the idea of augmenting the training data set with additional musical knowledge from the user.

**III. To design a system for designing musical agents that retains the benefits of programming by example and satisfies the requirements listed in Section 1.6.1, while removing the need for large amounts of training data**

**III (i). To identify the machine learning techniques to underly the system:** We developed the Agent Designer Toolkit (ADTK), which is software for designing musical agents based on a set of example performances. Thus, it also supports the paradigm of programming by example. However, its use involves significant design activities, in which the user configures machine learning algorithms and chooses *custom variables* which are salient features of the training data, in order to compensate for having very few training examples.

The ADTK is used to produce models of arrangement-level musical decision making that comprise (i) variable order Markov models, (ii) association rules, and (iii) custom variables (user-defined features). Variable order Markov models (VMMs) and association rules were chosen for their complementary modelling capabilities and also out of usability considerations: models relating to each can be found in various popular music software packages, so we envisaged that musicians would be able to fruitfully engage with them. However, we found that using only combinations of VMMs and association rules it was not possible to model certain important musical patterns. Thus, we added a design phase in which the user selects custom variables: salient features of the training data to make the modelling more effective.

**III (ii). To implement a real-time decision maker that can use the learnt models to take part in musical performance:** Before these models could be evaluated, it was required to develop an additional component to run them in real time. The association rules and custom variables are essential for effectively modelling musical
patterns, but they are not generative models. The combination of association rules and the relationships imposed by custom variables comprises a constraint satisfaction problem (CSP). However, it is generally not possible to predict in advance, how long it will take to solve a CSP. Therefore, in our method the CSP is transformed into an equivalent binary decision diagram, which is a data structure with special properties that allow certain CSPs to be solved in predictable lengths of time. We showed that for a selection of the most musically successful agents presented in this thesis, the decision making calculation can be carried out in consistently less than 60 ms on a standard laptop computer (usually much less than this). This novel real-time decision maker may have implications for other interactive music systems that incorporate CSPs.

III (iii). To evaluate the modelling capabilities of the system: Once the ADTK software had been implemented, the first question that we asked is whether it was capable of modelling a sufficiently wide variety of arrangement-level musical decision making behaviours. To answer this, we conducted the study described in Chapter 7, in which agents were designed to emulate the performance styles of practising electronic musicians playing particular compositions of their own. We empirically showed that in most cases, we were able to use the software to create agents that could convincingly emulate performance styles to standards in the region of those required for public performance: five of seven participants indicated that they had at least 70% confidence that their preferred agent could do a satisfactory performance (of the particular composition that they had demonstrated) in their stead. Related to this, we found, albeit from a small amount of data, that participants’ confidence that an agent will do a satisfactory performance correlates better to their judgements of its musicality, than to their judgements of how good it is at style emulation.

III (iv). To evaluate the system with respect to incorporating it into the workflow of contemporary computer music practitioners: The second question that we asked was how well the ADTK would support the design of musical agents
in real-world experimental music contexts. (Experimental music most commonly provides venues for performances with musical agents.) To address this, we carried out the five case studies of collaborative design of interactive and generative systems, described in Chapter 8. We showed that, at least when the user is familiar with the software, it can be used to develop agents for a variety of contexts and often, agents can be developed very quickly which is a huge advantage in collaborative, bricoleur-style system development.

In addition, the experiences of the design process that we reported (and triangulated with data from various other sources) showed some of the shortcomings of auditioning alone as a method for gaining an understanding of the model of musical performance embodied by an agent. Subjectively, the process was fatiguing, however, probably more generally it is simply not that easy to do, particularly as models get more complex. We made a two concrete proposals for ways of visualising the model embodied by an agent in order to aid this process: displaying visualisations of a range of agent performances, generated opaquely, and displaying statistics about these performances side-by-side with the same statistics calculated from the example performances. These two features could easily be added to the software.

III (v). To assess the usability of the system by non-programming end users: Our experiences of the design process reported in Chapter 8 hinted at some of the usability issues of the ADTK. In Chapter 9 we studied the usability of the software more rigorously. We began with a theoretical analysis using the Cognitive Dimensions of Notations. This showed that the presence of hidden dependencies, the requirements for users to master numerous abstractions, the need for visibility and juxtaposability, and the lack of role-expressiveness were key issues to be addressed to improve the usability of the ADTK. This was supported by the results of a preliminary usability study in which a participant successfully used a variety features of the ADTK to achieve certain musical goals; where difficulties arose, they were due primarily to hidden dependencies between various components of the system.
11.1.1 Supplementary Findings Relating to the ADTK

The study reported in Chapter 7 resulted in a number of findings that will inform the future development of the ADTK. In Section 7.4.1 we characterised musical decision making behaviours that were difficult to model with the ADTK. The most important of these relate to musical structure on a macro time scale, that is, where there are distinct musical sections characterised by different sets of statistics. We proposed augmenting the software to allow multiple agents to be trained using different user-defined regions of the training data set (i.e. different musical sections: one agent per section). This would be a straightforward change to make to the software and there are a number of simple and promising methods in the literature for implementing the scheme by which the agents themselves would be sequenced during a performance.

Additionally, we characterised a number of preset learning configurations (i.e. configurations of the model parameters and custom variables). Preset learning configurations would allow the ADTK to be used largely as a black box for agent design. None of the presets we evaluated in the study were universally effective, however the results did support the notion that a selection of presets could be arrived at, that would be reasonably effective in a wide range of musical contexts. Such a selection could form the basis of a high-level interface for the ADTK by which a user could completely avoid having to engage with the specifics of the learning configuration. We made a methodological recommendation that the study be iterated in order to develop and refine the selection of presets.

Finally, we identified of a number of common modelling sub-problems (i.e. specific musical patterns to be modelled) and techniques for capturing them effectively in ADTK agents. These include modelling overall dynamic structure in music; effectively capturing hypermetrical structure; and incorporating into a model when certain groups of musical elements are interchangeable, in order to lead to greater variety in an agent’s performances. We proposed that these modelling techniques could form the basis of the intermediate level of a multilayered interface for the
ADTK that would allow these common patterns to be individually selected by a user.

### 11.2 Final Remarks

Discussing her use of Max Mathews’ GROOVE system [125] in the 1970s, computer music pioneer Laurie Spiegel describes

‘using logic to automate the parts of the compositional process that could be automated, and leaving myself free to do the things that I can’t automate because I don’t understand why I’m doing them’ [131].

Spiegel’s work with GROOVE was a major milestone in the then nascent field of real-time computer music. Since that time, it has become possible not only to automate musical actions but to automate complex musical decision making. However, the idea remains the same: not to replace human creativity, but to support and supplement it.

Spiegel’s use of GROOVE was part of another beginning in the field of computer music: that of the democratization of computer music technology, which began with fortunate artists being given access to expensive and exclusive technology, and progressed as that technology became cheaper and more commonplace. Now, the ongoing challenge is not to make the technology less costly, but to make it more accessible to more people. In 2003, Shneiderman remarked that while ‘techno-utopians believe that computing technology has steadily improved over four decades, the fact remains that it is . . . too difficult to use’ [159]. In the decade since, computer music software has progressed with powerful new features for sound synthesis and signal processing but the interfaces have remained largely the same.

In this thesis we have developed methods and tools to open the design of musical agents to a much greater number of musicians. The Agent Designer Toolkit represents a marked evolution from GROOVE in its support of the design of real-time musical decision making, in its compatibility with low-cost hardware and in its
promise of accessibility to musicians without expertise in conventional computer programming. These characteristics will doubtless improve and be further proven, as our future work, for which we have laid the necessary foundation, comes to fruition.
Bibliography


In *Proceeding of the seventh ACM conference on Creativity and cognition*, Berkeley, CA, October 2009. (cited page 355)


[82] A Freed. Features and future of open sound control version 1.1 for NIME.


[91] M Hamanaka, M Goto, H Asoh, and N Otsu. A learning-based jam session system that imitates a player’s personality model. *Proceedings of the 18th*


[103] LP Kaelbling, ML Littman, and Anthony Cassandra. Planning and acting in
partially observable stochastic domains. Artificial Intelligence, 101(1-2):99–134,
1998. (cited page 76)

[104] Kevin Karplus and Alex Strong. Digital Synthesis of Plucked-String and Drum

(Popular) Music. In Australasian Computer Music Conference, pages 37–42,
Canberra, Australia, 2010. (cited page 47)


[107] Andrew J Ko, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw,
Susan Wiedenbeck, Robin Abraham, Laura Beckwith, Alan F Blackwell, Mar-
garet M Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, and Henry
Lieberman. The state of the art in end-user software engineering. ACM Comput.

Press, 1990. (cited pages 70, 71, and 74)


[110] Kyogu Lee and Malcolm Slaney. Acoustic Chord Transcription and Key Ex-
traction From Audio Using Key-Dependent HMMs Trained on Synthesized
Audio. IEEE Transactions On Audio Speech And Language Processing, 16(2):
291–301, February 2008. (cited page 69)


[177] K Vredenburg, JY Mao, and PW Smith. A survey of user-centered design


Appendix A

Using the Agent Designer Toolkit

The two main components of the Agent Designer Toolkit are the Agent Designer and the Fast Performer. The Agent Designer is used to configure the machine learning algorithms to train agents and the Fast Performer is used to audition them.

The Agent Designer is implemented both as a standalone program and as a plug-in for the Max interactive platform. In both cases, the user interacts with it primarily via its GUI. In contrast, the Fast Performer has only been implemented as a plug-in for Max. Using the plug-in forms of the two components, it is possible to design and audition agents intended to perform with any suitable system (i.e. feature extractor and generator) implemented in Max. Furthermore, since Max offers extensive facilities to communicate with other software and hardware, it is possible to design agents for many other scenarios too. Finally, the Agent Designer and Fast Performer can also be embedded in Ableton Live using a third component of the ADTK called the Agent Designer Device. This allows the development of agents for performing with musical material composed in Ableton Live. In this appendix, we describe the use of the Agent Designer GUI to create a learning configuration (Section A.1), the use of the ADTK in Max (Section A.2), and lastly, its use in Ableton Live (Section A.3).
Appendix A. Using the Agent Designer Toolkit

A.1 The Agent Designer GUI

Upon loading, the main Agent Designer window comprises a top panel with four buttons that are always available to the user, a main panel containing a tabbed interface of five tabs (Examples, Variables, Groups, Learn and Agent), and a status bar to display notifications to the user (see Figure A.1).

A.1.1 The Top Panel

The four buttons of the top panel are as follows. The Load Session and Save Session buttons allow Agent Designer session files to be loaded and saved, respectively. An Agent Designer session file is a binary file with the extension .agtk and it stores the training data and learning configuration required to train an agent. The Learning Configurations button allows preset learning configurations to be loaded. Finally, the About button displays a window containing information about the software.
A.1.2 The Examples Tab

With the Examples tab selected, the top of the main panel shows buttons that allow new examples to be imported. The Import CSV file button allows an example to be imported from a comma-separated-value (CSV) file. The Add example from RAW file button allows an example to be imported from a text file with formatting that is particular to the Agent Designer.

The remainder of the main panel shows the details of the examples that comprise the training data set of the current session. If there are no examples, the panel shows only the message ‘No examples’, (as in Figure A.1), otherwise a list of the examples is displayed (Figure A.2). Each example has two attributes associated with it. The first is the Active flag that indicates whether the example should be included in the training data set (it can be useful to temporarily omit an example during certain phases of the design process). The second attribute is the Loop flag that determines whether ‘looped’ sequences will be used to train the VMMs (see Section 2.3.2). This is checked by default and there is rarely a need to modify it. Finally, each example has associated with it an Edit button and a Delete button. The latter removes the example from the session and the former opens an Edit Example sub-window, in which the data can be examined and edited.

The Edit Example window (Figure A.3) shows the data associated with a particular example. Each row corresponds to a variable, and each column, to a decision point. Individual values can be changed, and columns (i.e. decision points) can be added and deleted. The example can also be exported as a CSV file.

A.1.3 The Variables Tab

When the Variables tab is selected, details of the variables and custom variables can be examined and edited (Figure A.4). The main panel displays a table with seven columns. The first is a unique integer to identify each variable (set by the software), and the second is a user-editable label which is not required to be unique. The third
Figure A.2: Screenshot of the Agent Designer Examples Tab. The list of examples is shown, as well as attributes (Active and Loop) for each one.

Figure A.3: Screenshot of the Agent Designer Example Edit window. The custom variable values are displayed in rows below the recorded values. However, only the recorded values can be edited manually.
column indicates whether each variable is categorical or ordinal. This must be set by the user for variables other than custom variables. Variables corresponding to Ableton Live tracks, for example, are categorical since there is usually no natural ordering of the clips. However, the feedback parameter of an echo effect is ordinal, since the values do have a natural ordering. The fourth column indicates whether the variable is an input variable, meaning that it is musician-controlled; an output variable, meaning that it is to be controlled by the agent; or an internal variable, which is synonymous with a custom variable. The fifth column is labelled Output Freq and it governs the frequency with which the values of output variables are actually output by the agent. By default, during a performance, values are only output when they change, but in some circumstances it is useful to force output at regular intervals. The Modeling column displays which variables are being modeled by (i) uniform random distributions, or (ii) VMMs. In the latter case, the maximum order allowed for the VMM is displayed. It is possible to apply models only to output and internal variables. The final column, labelled Priority, shows the user-specified priority for each variable to which a model has been applied.

In addition to the table showing the variables and custom variables, there are buttons to add, edit and delete custom variables. The Add Custom Variable and Edit Custom Variable buttons open a dialog box for choosing the properties of a custom variable (Figure A.5). This allows the name of the custom variable to be set; its type, via a drop-down menu; its critical value, if needed; and finally the selection of variables on which the custom variable depends. The software automatically removes any variables from the selection that would cause a circular dependency to arise.

### A.1.4 The Groups Tab

When the Groups tab is selected, the main panel displays the user-specified rule groups (Figure A.6). In the figure, there are three groups, High Feedback rule, Example Rule and All. Each group has a single boolean attribute, the Active flag, which
Figure A.4: Screenshot of the Agent Designer Variables Tab. The attributes for each music system variable custom variable are displayed.

Figure A.5: Screenshot of the Agent Designer Add/Edit Custom Variable Dialog Box. The type and critical value can be set, in addition to the set of underlying variables on which the custom variable depends.
indicates whether the group should be included during in the ARL analysis. At the bottom of the main panel are buttons to add, edit and remove rule groups.

The Add New Group and Edit Group buttons are used to open the Add/Edit Rule Group dialog box (Figure A.7). This allows the user to specify the name of the group; its minimum support; its minimum confidence; its maximum item set size; and the variables that it contains.

### A.1.5 The Learn Tab

When the Learn tab is initially selected, the main panel shows a button labelled Learn and an empty text area. Upon clicking the Learn button, the various learning algorithms are run and a summary is printed to the text area, comprising a list of the variables, their attributes, and the rules resulting from the ARL analysis (see Figure A.8).
Appendix A. Using the Agent Designer Toolkit

Figure A.7: Screenshot of the Agent Designer Add/Edit Rule Group Dialog Box. The variables to be included in the rule group are selected, along with the minimum support for discovered rules (S), the minimum confidence (C), and the maximum item set size (#).

Figure A.8: Screenshot of the Agent Designer Learn Tab.
A.1.6  The Agent Tab

Finally, when the Agent tab is selected, the main panel shows a text area into which notes about the agent can be entered, and a Save Agent button that allows the agent file to be saved for loading by the Fast Performer.

A.2  The ADTK in Max

The GUI just described is used for interacting with the Agent Designer both as a standalone program and as a plug-in. Next, we describe the use of the Agent Designer and the Fast Performer in the Max environment.

A.2.1  The Agent Designer and Fast Performer Max Externals

The ADTK includes two Max externals. The first is the Agent Designer external. This was implemented in Java and has its own GUI, that appears in a separate window. This GUI was described in Section A.1. The second Max external in the ADTK is the
Fast Performer. It was implemented in C++ and it has no GUI; it is interacted with exclusively through the mechanisms provided by Max.

It is straightforward to use the Agent Designer and Fast Performer in a Max patch. Figure A.10 shows a simple patch that allows (i) a training data set to be recorded, (ii) an agent to be designed and (iii) then used to control a set of four music system variables. The variables are controlled in the patch using the user interface elements coloured blue: two toggle switches, a four-way selector (known in Max as a radiogroup) and a slider. To record an example, a user clicks the Start Recording button, and then manipulates variables using the blue user-interface elements. At regular intervals, a snapshot is stored of the values of the four music system parameters. Once the Stop Recording button is clicked, the recording stops and a new example appears in the Agent Designer GUI (not shown in the Figure). Once some examples have been recorded, an agent can be created using the Agent Designer, and then loaded into the Fast Performer external.

Once an agent has been loaded, the Fast Performer can be activated. Then, at regular intervals, it outputs new values for the four music system variables according to the behaviour specified in the .agent file. Various messages can be sent to the Fast Performer in order to clear its memory (i.e. reset the VMMs so the histories have length zero), load new agent files, or query its current configuration.
Figure A.10: Screenshot of a Max patch incorporating the ADTK. (a) The top part of the patch includes the user interface elements for controlling four parameters of a notional music performance system, in addition to the Agent Designer external. The orange open button is used to open the Agent Designer GUI, presented in Section A.1, while the Start Recording and Stop Recording buttons are used, respectively, to start and stop recording example performances. (b) The lower part of the patch includes the am.fp external which can load agent files and generate parameter control data.
Appendix A. Using the Agent Designer Toolkit

A.3 The ADTK in Ableton Live

The Agent Designer device is a Max for Live device that encapsulates the Agent Designer and Fast Performer Max externals. It is loaded into a Live set as a plug-in, and can then be used to record a training data set, to design agents, and to use agents to control the Live set (see Figure A.11). Agents created in this way can control both the sequencing of clips in the Live set, and the values of discrete (i.e. integer-valued) parameters associated with software instruments and effects.

To create a training data set, the following steps are taken (refer again to Figure A.11). First a selection is made of the parameters of software instruments and effects that the agent will control. This is done using a sub-window opened using the Open Parameter Window button in the Agent Designer panel. The parameters associated with sequencing clips are automatically included in this group, so they
do not have to be selected explicitly. Then, recording is enabled from the Record
Controls panel and a performance is carried out as normal. When the performance
is stopped, a new example will be visible in the Agent Designer window (opened
using the Open Agent Designer button) and further examples can then be recorded.

The agent design process is conducted using the Agent Designer GUI as described
above. To audition agents, controls in the Agent panel are used. It is possible to load,
activate and clear agents, as well as to reset their histories. The Tries parameter in
the Agent panel is related to the procedure used by the Fast Performer component to
choose new parameter values (the parameter $N$, described in Section 6.2).
Appendix B

Association Rule Learning

Here we give a brief synopsis of association rule learning and some related algorithms. In doing so, we draw substantially from [92, Chapter 14], and the reader is referred to this text for a more detailed exposition.

Association rule learning (ARL) is concerned with finding joint values of the variables \( X = (X_1, X_2, ..., X_p) \) that occur frequently in a data set. Let the domain of a particular variable, \( X_j \), be denoted \( S_j \), and let \( s_j \) be a subset of this domain (i.e., \( s_j \subseteq S_j \)). Then, the general goal of ARL is to find subsets, \( s_1, ..., s_p \), such that the following probability is high:

\[
P \left( \bigcap_{j=1}^{p} (X_j \in s_j) \right). \tag{B.1} \]

This intersection of subsets is known as a conjunctive rule. Where \( s_j = S_j \), the variable \( X_j \) is said not to appear in the rule (i.e. the variable \( X_j \) can take any value in its domain).

This type of analysis has been used in marketing to find supermarket items that are commonly bought together. To perform such market basket analysis, a training data set is constructed in which each data point describes the contents of a shopper’s
basket at a point of sale. The domain of each variable, $X_j$ is binary ($X_j \in \{0, 1\}$) and $j$ corresponds to a particular item for sale (i.e. $X_j = 1$ means that item $j$ was in the shopper’s basket, and $p$ is the number of items for sale in the shop). Thus, each conjunctive rule found will describe a set of items that are commonly bought (or not bought) together.

The Apriori algorithm [3] provides an efficient way of solving a constrained version of the problem given in B.1. It requires that the data set contains only variables with finite domains (discrete-valued variables). The variables are encoded using the technique of dummy variables as a set of binary valued variables, $Z = Z_1, ..., Z_K$, where

$$K = \sum_{j=1}^{p} |S_j|,$$

so each variable, $X_j$, is encoded using $|S_j|$ binary variables. An association rule is found when subset, $\mathcal{K}$, of the integers $1, ..., K$ is identified such that the probability,

$$P \left( \bigcap_{k \in \mathcal{K}} (Z_k = 1) \right),$$

is high. In effect, this is the same as B.1 except that the cardinalities of the subsets $s_j$ are constrained to unity.

In the terminology of ARL, the subset, $\mathcal{K}$ is known as an item set and an item set has two important attributes. First its size is the number of variables, $Z_k$ that it contains. Second, its support, denoted $T(\mathcal{K})$, is the number of times it appears in the data set, expressed as a fraction of the total number of data points in the data set. The Apriori algorithm finds all item sets, $\mathcal{K}_i$, with $|\mathcal{K}_i|$ less than or equal to a maximum size, and with support greater than or equal to a minimum support value. Both the maximum size and minimum support are chosen by the user.

The high-support item sets can be used to derive association rules, which are implies rules of the form

$$A \implies B$$

(B.4)
where $A$ and $B$ are disjoint subsets of an item set $\mathcal{K}$. The item set $A$ is known as the antecedent and $B$ is known as the consequent of the rule. An example of such a rule as might be relevant to arrangement-level musical decision making is:

\[
\text{Bass Drum} = 1 \ \text{AND} \ \text{Snare Drum} = 0 \ \Rightarrow \ \text{Hi-Hat} = 0, \quad (B.5)
\]

meaning that when the bass drum is sounding but the snare drum is not, then the hi-hat must not sound either.

In addition to the support and size of the item set, $\mathcal{K}$, from which it is derived, an association rule also has a confidence attribute associated with it. The confidence, $C(A \Rightarrow B)$ of a rule, $A \Rightarrow B$, is the fraction of times the rule is correct, given by

\[
C(A \Rightarrow B) = \frac{T(\mathcal{K})}{T(A)}. \quad (B.6)
\]

Thus, to search for association rules using the Apriori algorithm, it is necessary to specify (i) the maximum item set size, (ii) the minimum support, and (iii) the minimum confidence.