User Hints for Optimization Processes

by

Hugo Alexandre Dantas do Nascimento


A thesis submitted to
The School of Information Technologies
The University of Sydney
for the degree of
DOCTOR OF PHILOSOPHY

November, 2003
I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Hugo Alexandre Dantas do Nascimento

Sydney
10th November 2003
To my parents, for their love and motivation,
and to my daughter.
Acknowledgements

This thesis would not have been completed without the help of many people. I would like to express here my immense gratitude to all of them.

Prof. Peter Eades, my supervisor, persuaded me to investigate the field of Human-Computer Interaction, and provided a dynamic and enthusiastic environment for my research. Peter’s talented way of solving problems also had a great effect on my view of the world; he set an example of dedication and commitment to scientific research that I want to follow in life.

My parents, my sisters and my girlfriend were with me all the way, despite the geographical distance. By phone and mail they cared for my health and studies, and motivated me to look after myself. One of my sisters, Hani, also came to Australia and spent almost a year with me. Their continuous support was important to the conclusion of this work.

During these four years of research I received a Ph.D. scholarship from CAPES-Brazil. I also had a salary from the Universidade Federal de Goiás – Brazil, where I hold a lecturer position. Without this financial support, covering my living expenses and the tuition fees of the university in Australia, my stay here would not have been possible. This is the kind of investment that really brings positive results to a nation, and I am pleased to see that these two institutions did it well. Furthermore, I want to thank the Australian Research Council for providing me with an annual research allowance and for supporting my participation in conferences.

Several researchers made valuable comments about the topics that I investigated. I would like to thank Dr. Seokhee Hong and Dr. Masahiro Takatsuka (from the School of Information Technology at The University of Sydney), and the international researchers with whom I had interesting discussions. I am also in debt to all the cartographers that participated in the evaluation of our prototype for the Map Labeling problem. These include Murray Godfrey, David Godfrey and Yan Tollis (from Sydway/Ausway), Graham Russel and his colleges (from UBD), and the team of experts under the supervision of Brian McLauchlan (from DIGO-Australia).
My thanks go as well to the administrative and technical staff of the University of Sydney, and of
the University of Newcastle (where I spent my first year of studies before Peter Eades moved to
Sydney). At the University of Newcastle, the administrative support of Diana Edwards, Debbie
Hatherell, Aaron Scott and David Montgomery provided all help that I needed. I received similar
support at The University of Sydney from Sharon Chambers and Josephine Spongberg, and from
the “workshop guys” – Remo Di Giovanni, Arthur Scott, Witold Janus, Robert Calabrese and Allan
Creighton. I also want to thank Eu Wee Tan, who worked as a research assistant and helped me to
organize the human experiments with a genetic algorithm.

Very good friends shared their time with me in Australia. When I needed a break from my studies
I used to socialize with them. This always refilled my motivation to work. I cannot name all these
friends here, but certainly a few ones have to be mentioned: Tony and Dirce Mendonça, Des and
Dawn Mcmeekin, Hui-Ju Cheng (Jessica), Bernd Scheumman, Anja Kemena, Uta Zander, Kim
Wellard, Luzenira Brasileiro and Clerilei Bier.

Finally, I am grateful to Stephen Anthony, Tim Dwyer, Chris Ottrey and again to my supervisor and
to Dr. Seokhee Hong for having reviewed this thesis.
# Contents

1 Introduction

1.1 Motivation ........................................... 1
1.2 Aims ...................................................... 5
1.3 Research Methodology ................................. 6
1.4 Contributions ........................................... 7
1.5 Organization of the Thesis ............................ 8

2 Background

2.1 Basic Concepts ......................................... 10
2.1.1 Graphs .................................................. 10
2.1.2 Combinatorial Optimization ......................... 11
2.1.3 Human-Computer Interaction and Usability Studies ............. 17
2.1.4 Information Visualization ............................ 20
2.2 Human Interaction in Related Areas .................... 21
2.2.1 Computer Aided Design .............................. 23
2.2.2 Systems for Information Retrieval ..................... 28
2.2.3 Mixed Initiative Systems ............................. 30
2.3 Interactive Optimization ................................. 31
2.3.1 Interactive Evolutionary Approaches ................... 32
2.3.2 Haptic Hints .......................................... 34
2.3.3 Human-Guided Search ............................... 35
2.3.4 Other Approaches .................................... 38
2.4 Summary of Human-Computer Collaboration ............. 41

3 The User Hints Framework ............................... 44
3.1 The Elements of the Framework ....................... 44
3.1.1 Types of Hints ........................................ 46
3.1.2 Constraints and Feasibility ......................... 47
3.1.3 The Initial Solution .................................. 47
3.1.4 Quality Function .................................... 48
3.1.5 Optimization Methods ............................... 48
3.1.6 The Visualization Tool .............................. 49
3.1.7 Work Modes ......................................... 50
3.2 Comparison with Other Approaches .................. 51

4 User Hints for Graph Clustering .......................... 54
  4.1 Graph Clustering ...................................... 54
  4.2 An Interactive Framework for Graph Clustering ... 56
  4.3 The ClusterHints System ............................. 59
  4.4 Remarks ............................................... 64
    4.4.1 Related Work ..................................... 67
    4.4.2 General Conclusions .............................. 68

5 User Hints for Directed Graph Drawing .................. 70
  5.1 Graph Drawing ........................................ 70
  5.2 User-Interaction Suitability ......................... 72
  5.3 An Interactive Graph Drawing Framework .......... 73
    5.3.1 The Sugiyama Method ........................... 75
    5.3.2 Implementing User Hints in the Sugiyama Method ... 76
  5.4 The GDHints System .................................. 81
  5.5 Pilot Study ........................................... 83
    5.5.1 Experiment Setup ................................ 83
    5.5.2 Results .......................................... 85
  5.6 Remarks ............................................... 90
    5.6.1 Constraints ...................................... 90
    5.6.2 Focus ............................................ 91
    5.6.3 User Search Method ............................. 91
    5.6.4 Quality Feedback ................................. 92
    5.6.5 Task Division ................................... 92
### 5.6.6 Properties of the Greedy-Cycle-Removal Heuristic .......................... 92
### 5.6.7 Related Work ....................................................................................... 97
### 5.6.8 A Better Optimization Method in the GDHints System ...................... 98

### 6 A Focus and Constraint-Based Genetic Algorithm ................................. 99

6.1 Genetic Algorithms for Graph Drawing ................................................. 99
6.2 Graph Drawing Definitions ..................................................................... 101
6.3 The Genetic Algorithm ......................................................................... 102
   6.3.1 Individuals ...................................................................................... 103
   6.3.2 Quality Evaluation ......................................................................... 105
   6.3.3 Evolutionary Cycle .......................................................................... 106
   6.3.4 Operators ....................................................................................... 108
6.4 Integration into the GDHints System ...................................................... 111
6.5 Evaluation .............................................................................................. 113
   6.5.1 Experiment Setup ........................................................................... 113
   6.5.2 Results .......................................................................................... 115
6.6 Remarks .................................................................................................. 144
   6.6.1 Focus and Constraints ................................................................... 144
   6.6.2 Human Interaction ......................................................................... 145
   6.6.3 The Internal Structure of the Genetic Algorithm ......................... 147
   6.6.4 Hill Climbing ................................................................................... 149

### 7 User Hints for Map Labeling ................................................................. 154

7.1 The Map Labeling Problem and Automatic Methods ........................ 154
7.2 The Need for User Intervention ............................................................. 159
7.3 Labeling Steps ....................................................................................... 161
7.4 An Interactive Map Labeling Framework ............................................ 164
   7.4.1 Interactions ..................................................................................... 166
   7.4.2 Solution Quality ............................................................................. 168
   7.4.3 Optimization Methods and Focus ............................................... 168
   7.4.4 Visualizations ............................................................................... 170
   7.4.5 Labeling Improvement ................................................................ 172
   7.4.6 Selection Extension ................................................................. 174
B.4 The LabelHints System ............................................. B-12
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Duration and costs of flights between cities.</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Post-processing improvement of a solution for an optimization problem.</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>A general diagram of the User Hints framework.</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>The Simulated Annealing algorithm.</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>3D plot of a mathematical function.</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>A colored map indicating fire seasons in Australia.</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>Parallel Coordinates.</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>The beautification of a drawing.</td>
<td>25</td>
</tr>
<tr>
<td>2.6</td>
<td>GLIDE: an interactive constraint-based system for drawing graphs.</td>
<td>26</td>
</tr>
<tr>
<td>2.7</td>
<td>Examples of Design Galleries interfaces.</td>
<td>27</td>
</tr>
<tr>
<td>2.8</td>
<td>SMILE: a Design Gallery system for Graph Drawing.</td>
<td>27</td>
</tr>
<tr>
<td>2.9</td>
<td>Interactive Evolutionary Computation where the user performs the fitness function.</td>
<td>33</td>
</tr>
<tr>
<td>2.10</td>
<td>Examples of problems for the Haptic Hints framework.</td>
<td>35</td>
</tr>
<tr>
<td>2.11</td>
<td>The HuGSS system for the Capacitated-Vehicle-Routing-with-Time-Windows problem.</td>
<td>37</td>
</tr>
<tr>
<td>2.12</td>
<td>The Interactive Genetic Algorithm (IGA) approach.</td>
<td>39</td>
</tr>
<tr>
<td>2.13</td>
<td>The Kaleidoscope Visualization.</td>
<td>40</td>
</tr>
<tr>
<td>2.14</td>
<td>User actions classified according to the two major goals.</td>
<td>43</td>
</tr>
<tr>
<td>3.1</td>
<td>The User Hints framework.</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>The elements and the steps involved in the User Hints framework for Graph Clustering.</td>
<td>58</td>
</tr>
<tr>
<td>4.2</td>
<td>The ClusterHints system for Graph Clustering.</td>
<td>60</td>
</tr>
<tr>
<td>4.3</td>
<td>Three visualizations of a clustering: a histogram chart, a scatter-plot graphic, and a graph drawing.</td>
<td>65</td>
</tr>
</tbody>
</table>
4.4 A visualization of a graph partitioning using Springs. .......................... 67
5.1 The general graph drawing framework. ............................................. 71
5.2 A random and a spring-based drawing of the graph structure of the Information Visualization Group web site and related pages. ......................... 71
5.3 The interactive framework for Graph Drawing. ................................. 75
5.4 Different ways of drawing a cycle. .................................................. 77
5.5 The modified version of the Greedy-Cycle-Removal heuristic. .......... 78
5.6 The approach for Layer Assignment with support to constraints and focus. ........................................................................... 79
5.7 The barycenter algorithm for Crossing Reduction with support to constraints and focus. .......................................................... 80
5.8 The \texttt{FixConstraints} heuristics. ............................................... 80
5.9 GDHints – an interactive Graph Drawing system based on user hints. . 82
5.10 Changes in the number of edge crossings caused by user actions. .... 87
5.11 Drawings of the predator-pray ecologic system graph. ...................... 88
5.12 Drawings of the C language syntax graph. ....................................... 89
5.13 Drawings of the Unix family-tree graph. ......................................... 89
5.14 Drawings of the Forrester’s World Dynamics Diagram. .................... 89
6.1 Drawing of a directed graph on a grid. ............................................. 102
6.2 Dependence between selected and fixed elements of a graph when redrawing selected vertices. ............................................................... 103
6.3 Representation of an individual in the genetic algorithm. .................. 104
6.4 Internal functioning of the genetic algorithm. ................................... 107
6.5 Improvement of a selected graph drawing region. ............................. 112
6.6 Simultaneous improvement of two disjoint selected graph drawing regions. ........................................................................ 113
6.7 Drawing of the \texttt{Worlddy} graph with a promising area for improvement. ................................................................ 118
6.8 Drawings of the \texttt{Csyntax} graph in experiment \texttt{A1}. .................... 135
6.9 Drawings of the \texttt{Klayer} graph in experiment \texttt{A2}. ......................... 135
6.10 Drawings of the \texttt{Unixsys} graph in experiment \texttt{A3}. ....................... 135
6.11 Drawings of the \texttt{Worlddyn} graph in experiment \texttt{A4}. .................... 136
6.12 Drawings of the \texttt{Knation} graph in experiment \texttt{A5}. ....................... 137
6.13 Drawings of the \texttt{Telcall} graph in experiment \texttt{A6}. ....................... 138
7.22 An optimization task where several labeling problems need to be solved. . . . . . 185
7.23 Feature selection running out of the screen area. . . . . . . . . . . . . . . . . . . . 186

8.1 A process for applying the User Hints framework to optimization problems. . . . 196
8.2 Levels of mobility for the Edge Crossing Minimization problem. . . . . . . . . . . 209

A.1 MERL’s Optimization Table. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . A-1
A.2 Our first optimization table with bottom-up projection. . . . . . . . . . . . . . . . . A-2
A.3 Our second optimization table with bottom-up projection. . . . . . . . . . . . . . . A-4
## List of Tables

5.1 Graphs used for the experiments with the GDHints system. ........................................... 84
5.2 Quality of the best drawings produced by the users for all graphs. ............................. 85
5.3 Overall results of the experiments compared to the quality of the initial drawings. ........ 86
5.4 Usage of the interactive tools of the GDHints System. .................................................... 87

6.1 Setup for the experiments with the genetic algorithm. ................................................... 114
6.2 Results of the experiment A1, for the graph Csyntax. .................................................... 122
6.3 Results of the experiment A2, for the graph Klayer. ...................................................... 123
6.4 Results of the experiment A3, for the graph Unixsys. ................................................... 124
6.5 Results of the experiment A4, for the graph Worlddyn. ................................................. 125
6.6 Results of the experiment A5, for the graph Knation. ................................................... 126
6.7 Results of the experiment A6, for the graph Telcall. ..................................................... 127
6.8 Results of the experiment A7, for the graph Gd94dir. ................................................... 128
6.9 Results of the experiment B1, for the graph Knation with the subjects performing only focus. .................................................. 129
6.10 Results of the experiment B2, for the graph Telcall with the subjects performing only focus. ................................................ 130
6.11 Results of the experiment B3, for the graph Unixsys with layout constraints. ............... 131
6.12 Results of the experiment B4, for the graph Worlddyn. ............................................... 132
6.13 Results of the experiment B5, for the graph Gd94dir. ................................................. 133
6.14 Details about the subjects and their total performance. ................................................. 134
6.15 Results produced by the Hill Climbing for 20 minutes. ............................................... 150
6.16 Results produced by the Hill Climbing for 40 minutes. ............................................... 151

A.1 Cost of the equipments for the second optimization table. ........................................... A-3
Abstract

Innovative improvements in the area of Human-Computer Interaction and User Interfaces have enabled intuitive and effective applications for a variety of problems. On the other hand, there has also been the realization that several real-world optimization problems still cannot be totally automated. Very often, user interaction is necessary for refining the optimization problem, managing the computational resources available, or validating or adjusting a computer-generated solution.

This thesis investigates how humans can help optimization methods to solve such difficult problems. It presents an interactive framework where users play a dynamic and important role by providing hints. Hints are actions that help to insert domain knowledge, to escape from local minima, to reduce the space of solutions to be explored, or to avoid ambiguity when there is more than one optimal solution. Examples of user hints are adjustments of constraints and of an objective function, focusing automatic methods on a subproblem of higher importance, and manual changes of an existing solution. User hints are given in an intuitive way through a graphical interface. Visualization tools are also included in order to inform about the state of the optimization process.

We apply the User Hints framework to three combinatorial optimization problems: Graph Clustering, Graph Drawing and Map Labeling. Prototype systems are presented and evaluated for each problem. The results of the study indicate that optimization processes can benefit from human interaction.

The main goal of this thesis is to list cases where human interaction is helpful, and provide an architecture for supporting interactive optimization. Our contributions include the general User Hints framework and particular implementations of it for each optimization problem. We also present a general process, with guidelines, for applying our framework to other optimization problems.
CHAPTER 1

Introduction

1.1 Motivation

With the technological revolution that marked the last century, there was a general feeling that technology would soon be able to automate almost all kind of activities performed by humans. The concept of a maid robot (such as Rosie, the maid robot in the cartoon show “The Jetsons”) is a representative example of people’s expectations at that time. In fact, during the last fifty years of the twentieth century, technology did replace humans in many activities, such as the jobs in the automobile industry [128].

However, a new century has begun, and many automatic tools expected to exist by this time (including Rosie) remain as intangible goals. In contrast to the technological predictions in the last century, we have now realized that several real-world problems are much more difficult than they seemed. Some of these difficulties appear in Artificial Intelligence, and in Image and Natural Language Processing; they are often associated with pattern recognition problems involving image, voice and video content. Another group of difficult problems comes under the heading of Combinatorial Optimization; this is the focus of this thesis.

A broad variety of techniques have been developed for Combinatorial Optimization problems. They include heuristic strategies dependent on the problem, Dynamic Programming [16], Integer Programming [84, 140, 168], and meta-heuristic methods – such as Greedy heuristics, Simulated Annealing [2, 107], Tabu Search [78, 79], Genetic Algorithms [81], GRASP [65, 66] and Asynchronous Teams [41, 42, 159, 181]. Although a considerable amount of research has been done

---

1 A small step towards a hard-worker and low-waged housekeeper has been made by researchers from MIT. It consists of a robot for cleaning floors called Roomba, which is available commercially since 2002. Information about Roomba can be found at http://www.roombavac.com/ and http://www.time.com/time/roomba/.

2 An optimization problem consists of finding the maximum or the minimum of a function defined on some domain. Furthermore, the solution of the problem usually has to satisfy a set constraints. The problem is said to be combinatorial if the domain if finite. A more detailed definition of a combinatorial optimization problem is given in Section 2.1.2.
in this area, many combinatorial optimization problems that have practical applications in the real world cannot yet be tackled satisfactorily via a fully automatic approach. Examples are bin-packing problems for the garment industry [85], steel, wood and glass cutting processes [15], vehicle routing problems [8], and timetabling [18, 83, 139, 197, 199]. The immediate reason for this lies in the computational complexity of these optimization problems. In the early 70’s, Cook [35], Karp [103] and Levin [125] formulated an important theory that defines some problems as belonging to classes called \(\text{NP-hard}\) and \(\text{NP-complete}\). This means that such problems are very difficult, and it is unlikely that they can be solved optimally in polynomial time\(^3\). As well as computational complexity, there are many other factors in the real world that contribute to the difficulty of a problem:

- The problem can be dependent on subjective domain knowledge. The domain knowledge may be difficult to express formally and may vary from person to person.

- Some characteristics of the problem may be unknown; in this case, the problem is potentially dynamic, since its objectives and constraints may need adjustment as the optimization progresses.

- The problem may include multiple objectives and constraints; optimizing only one of these aspects can be already an NP-hard problem; the challenge is then to consider all objectives and constraints simultaneously, and to find compromise solutions in conflicting situations.

- In general the problems studied in science are simplified models of real-world problems. As a consequence, most optimization methods available in the scientific literature do not handle all objectives and constraints that appear in practical cases.

- Finally, existing hardware technology (CPU power and memory size) may not be sufficient to deal with complex problems that contain several objectives or constraints, or have a large number of variables. Even when powerful computation resources do exist, they may be too expensive to acquire.

Complex problems, presenting one or more of the aspects above, are quite common in our daily life. Consider for example the job performed by a travel agent when an employee of a major company has to meet several customers in different cities in order to provide support or training for its products. The employee has to decide with the travel agent a flight itinerary that starts from his

\(^3\text{For more details about the theory of Computational Complexity, see Garey and Johnson [74].}\)
or her current location, passes by each of the customers’ cities, and finally returns to the starting point. The company usually specifies some basic criteria for guiding this decision process: the usual rules are that the trip has to be as cheap as possible, and that the employee should not stay away for a long period of time. If the time allocated for serving each customer is fixed, and if they can be visited in any order, then the problem consists of minimizing the duration and the cost of the flights. This involves a number of different flight options in the case that more than one airline provides routes between the customers’ locations; moreover, the airlines may have flights on different days of the week and with distinct time schedules. Figure 1.1 shows an example of a hypothetical network describing flights between six cities (labeled from A to F); each flight is represented by a line connecting two cities. The lines are labeled with the duration (in hours) and the cost of the flights. The duration represents the total flight time, from boarding to arrival and including intermediate stops in some cities not included in the diagram. Choosing the best itinerary for such a trip is in fact solving the Traveling Salesman Problem (TSP) with two objectives. The objectives are to minimize the sum of the costs and the sum of the durations of the chosen flights. The TSP is a well known problem, and is NP-hard even for a single objective. Satisfying two objectives simultaneously is a more difficult problem.

![Figure 1.1: Duration and costs of flights between cities.](image)

We can see that the problem to be solved by the employee and the agent is computationally difficult. Nevertheless, its entire complexity has not yet been discussed. Rather, we have only mentioned the company’s interests and the basic knowledge about the costs and durations of flights. Many other elements are commonly involved in flight itineraries. For instance, some airlines may offer special discounts if a return ticket or a package of two consecutive flights is bought. In addition, the employee may have special requests such as: to give preference to flights that include
1.1 Motivation

meals in the basic package, or to fly with an airline that has a specific frequent flyer program.

Conflicts can also exist between the objectives; for example, shorter flights may imply higher costs. Consequently several compromise solutions exist, and the employee may decide subjectively on which option to take. As an example of a subjective decision, some employees may prefer to save the company’s money by taking cheaper flights, which start early in the morning and have many connection stops.

Note that, if there is an emergency situation during the trip such that the employee has to prolong the stay in a particular city, then the itinerary should be amended without canceling all future flights.

Because of the enormous number of conditions and possibilities, many real-world combinatorial optimization problems still depend heavily on humans (in the “travel agency” problem above, the agent does most of the work). Computers can be used to search for possible initial solutions, but it is the human element that evaluates possibilities, recommends changes, and adjusts and approves the final solution.

In recent years, there has been an increasing interest in interactive tools for optimization methods. This is partially due to developments in User Interfaces, more generally in the area of Human-Computer Interaction, which now can provide intuitive and effective environments for interaction with a variety of applications. Human interaction in this context is beneficial since it provides a way of refining or adjusting the optimization problem to match the user’s desires, or of managing the computational resources available. There are also well-known differences between human and machine skills for problem-solving, which can be exploited by human interaction. Computers, for example, are suitable for intensive computation, where many solutions can be created and numerically evaluated. Humans, on the other hand, are skillful in identifying patterns that differentiate good from bad solutions.

The need for having humans involved in optimization tasks was discussed by Donald Edward Knuth during his lecture in the Graph Drawing Conference in 1996\(^4\), in Berkeley [145]. Knuth said:

“I also like to be able to tune things up later... I would urge all of you who are outputting the results of your graph drawing, not to just output a postscript file, but ideally you could output a file in a higher level language (and Metapost is the best I know), so that your users will be able to take that file and make slight refinements if they like afterwards, and rather easily.”

Later, when talking about drawings of tree structures, which are frequently used in his famous

\(^4\)A video of Knuth’s talk, “Graph Drawing from a User’s Perspective”, can be obtained from the Mathematical Science Research Institute, at Berkeley-CA, USA. See http://www.msri.org/index.html for more information.
1.2 Aims

In this thesis we investigate the issues raised by Knuth, and show human interaction can contribute to optimization processes. More precisely, we investigate the following questions:

- In what circumstances is human intervention necessary?
- Can human interaction be done during runtime in order to improve the optimization?
- What is the best architecture for achieving this goal?
- Are fully-automatic optimization methods still useful in an interactive environment? If so, what methods are more suitable and how can they be adapted to support interactive facilities?

For answering these questions we present an architecture that we call the User Hints framework. In our framework, users can control an optimization process by providing hints, which are changes to the objectives and constraints of the problem, and direct control of the optimization process. Such changes allow the users to include domain knowledge, escape from local minima, eliminate ambiguous situations or speed up the optimization process.

It is important to note that many optimization systems that support human interaction implement this feature as a post-processing step in a sequential approach. Figure 1.2 shows the traditional
framework with post-processing – which is also what Knuth has suggested: firstly an automatic method is applied, and then the human improves the computer-generated solution via a manual adjustment.

![Figure 1.2: Post-processing improvement of a solution for an optimization problem.](image)

The User Hints framework is different from the post-processing model, since it considers a stronger relation between the user, the automatic tool and the solution been improved. Figure 1.3 presents a general description of the User Hints framework. The automatic method acts as an improvement algorithm; not only can it be executed on the initial stage of the optimization (in order to produce a good initial solution), but can also be re-applied to improve solutions modified by the user. The users interact with the solution, the optimization method, and with the description of the problem. (Note that in the traditional post-processing framework the user can still return to the initial stage and change the problem or replace the optimization method; however, this implies a new optimization processing that does not take into consideration improvements done by the user on the previous solution.)

![Figure 1.3: A general diagram of the User Hints framework.](image)

We investigate the potential of the User Hints framework by applying it to some combinatorial optimization problems. We did not consider the “travel agency” problem. Instead, we approached three other more familiar problems: Graph Clustering, Graph Drawing and Map Labeling.

1.3 Research Methodology

The research methodology used during this work consisted of the following steps:

1. Investigating optimization problems that can benefit from human interaction.
1.4 Contributions

2. Defining a general interactive framework for optimization.

3. Building prototype systems to test interaction facilities. This involves:

   (a) building specific interactive frameworks;

   (b) building systems in the application domain.

4. Evaluating these systems using:

   (a) domain experts,

   (b) controlled experiments, and/or

   (c) quality parameters.

5. Refining the general interactive framework based on the experience with the systems, and developing guidelines for applying it to other problems.

1.4 Contributions

The objective of this thesis is to investigate whether human interaction can help to produce high quality solutions in optimization processes. We present a framework for this goal, and describe derivations of it for three case studies. The case studies show how user hints, automatic methods and visualization tools can be put together for Graph Clustering, Directed Graph Drawing and for Map Labeling. The main advantages and disadvantages of the User Hints framework arise from the experiments done with our prototype systems.

The specific contributions of the thesis are listed below:

- A general architecture for human interaction in optimization processes.

- An approach for including user hints in Graph Clustering. This was our first study of user hints and we applied the lessons learned from our experience with Graph Clustering in other case studies.

- An approach for incorporating user-defined layout constraints and focus with the Sugiyama method, for drawing directed graphs. Focus consists of restraining the scope of the method to work only on a selected part of the optimization problem.
1.5 Organization of the Thesis

- A new genetic algorithm for drawing directed graphs; this algorithm supports layout constraints and focus; it implements a design for the solution representation and for the evolutionary operators that yields good drawing solutions.

- For the Map Labeling case, an approach for focus that constructs a new labeling problem based on selected elements (features and labels) of the map.

- A recursive operation that computes the maximum set of elements of a cartographic map that may need to be relabeled in order to improve the label position of a particular feature.

- A process and intuitive guidelines for applying the User Hints framework to other optimization problems.

The results in this thesis can benefit researchers and system developers in the case study areas, as well as in major fields such as Human-Computer Interaction and Combinatorial Optimization. The discussions made throughout the thesis together with the guidelines given in the General Remarks Chapter provide sufficient material for implementing User Hints frameworks for other optimization problems. For the Graph Drawing and the Map Labeling problems, the thesis already includes a good interactive approach that can be used as a starting point for future extensions.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 provides a background on previous interactive approaches for optimization. Chapter 3 introduces the User Hints framework. Chapter 4 discusses the application of user hints to the problem of clustering graphs. Chapter 5 investigates how our framework can be used to improve drawings of directed graphs with the Sugiyama Method. Chapter 6 extends the graph drawing investigation by presenting a genetic algorithm that supports layout constraints and focus. Chapter 7 applies the User Hints framework to the problem of labeling point-features in maps. Chapter 8 discusses additional issues related to user interaction, presents a process and guidelines for applying our approach to optimization problems, and discusses extensions to the framework. Chapter 9 draws our general conclusions and proposes suggestions for future research.

The thesis also has two appendices: Appendix A presents two interactive tables that we built for experimenting with human interaction – the tables are based on the optimization table developed at
1.5 Organization of the Thesis

MERL, Boston [8]: Appendix B describes the content of the CDROM included with the thesis. It also explains the user interface of our prototype systems.
This thesis concerns user interaction with optimization processes. The thesis uses well established concepts in Graph Theory, Combinatorial Optimization, Human-Computer Interaction and Information Visualization; brief introductions to these areas are given in Section 2.1.

The remainder of this chapter describes examples of systems and approaches that involve collaboration between humans and computers working together to solve problems. Of course many thousands of systems involving human-computer collaboration have been designed since the advent of computers. We have restricted our attention to those which have some implications for the thesis.

In Section 2.2 we discuss human interaction in three general categories: Computer Aided Graphics Design, Information Retrieval and Mixed Initiatives.

In Section 2.3, we are more specific: we consider approaches and systems in which humans collaborate with optimization algorithms.

Section 2.4 brings the background together by summarizing the roles usually performed by humans and computers in interactive systems, and presenting two main goals for having human interaction in optimization processes. This leads to the development of the User Hints framework.

2.1 Basic Concepts

2.1.1 Graphs

A graph is a mathematical model widely used to describe relationships between entities. The definitions below follow the terminology used in [23, 179].

A (undirected) graph $G = (V, E)$ consists of a finite set of vertices $V$ and a finite set of edges $E$. An edge $e \in E$ is an non-ordered pair $(u, v)$ of vertices of $V$. We say that $u$ and $v$ in $V$ are

$^1$Also called general graph.
2.1 Basic Concepts

Adjacent or neighbors if there is an edge $e = (u, v) \in E$. In that case, we also say that $e$ is incident to $u$ and $v$, and that $u$ and $v$ are the endpoints of $e$. The degree of a vertex $u \in V$ is the number of edges in $E$ incident to $u$. A loop is an edge with one endpoint.

A path from a vertex $u$ to a vertex $t$ in $G = (V, E)$ is a sequence $(u = v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, (v_k-1, v_k), v_k = t)$ where $(v_i, v_{i+1})$ are edges of $E$ (for $i = 0, 1, \ldots, k - 1$) and $v_0, \ldots, v_k$ are vertices of $V$. Without loss of generality, we can omit the edges in the sequence. When a path has both extreme vertices the same ($v_0 = v_k$), it is called a cycle. The length of a path is the number of edges in the sequence. The (graph theoretic) distance between two vertices $u$ and $v$ in the graph is given by the shortest path from $u$ to $v$.

A graph $G$ is connected if there is a path between all pairs of distinct vertices $u$ and $v$ in $G$.

A graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. The subgraph $G'$ is an induced subgraph of $G$ if, for all $e = (u, v) \in E$ with $u, v \in V'$, $e \in E'$.

A tree is a connected graph that has no cycles.

Similarly, a directed graph $G = (V, E)$, consists of a finite set $V$ of vertices and a set $E$ of directed edges with ordered pairs of vertices of $V$. An edge $e = (u, v) \in E$ is said to be an outgoing edge from $u$ and an incoming edge to $v$. Note that the terms $(u, v)$ and $(v, u)$ define two different edges in a directed graph, while they represent the same edge in an undirected graph. For a vertex $w \in V$, we represent the number of incoming edges to $w$ by $\text{indeg}(w)$, and the number of outgoing edges from $w$ by $\text{outdeg}(w)$. The (total) degree of $w$ is $\text{indeg}(w) + \text{outdeg}(w)$. A vertex is a source if it has no incoming edge; it is a sink if it has no outgoing edge.

The concepts of path and graph connectivity are similar for both undirected and directed graphs.

2.1.2 Combinatorial Optimization

We use notations from [71, 86, 138] in this section.

In general, an optimization problem can be described as maximizing an objective function $f : D \rightarrow \mathbb{R}$ subjected to a set $C$ of constraints on $D$, where $D$ is the domain or search space of the problem. This definition describes a maximization problem; a minimization problem is similar. An optimization problem is multi-objective if it has two or more objective functions to be maximized/minimized.

The optimization problem is combinatorial if the domain $D$ is finite. The elements of $D$ are in general multidimensional vectors that represent variables of the problem. We often refer to the elements of $D$ informally as solutions of the problem.
2.1 Basic Concepts

Let $S \in D$ be the set of all vectors $x$ that satisfy the constraint set $C$. The problem is said to be feasible if $S \neq \emptyset$; otherwise, the problem is infeasible. All elements of $S$ are feasible solutions to the problem. An element $x \in D$ is infeasible if $x \notin S$. The global optimum or optimal solution is a solution $x^* \in S$ such that $f(x^*) \geq f(y)$ for all $y \in S$.

Many optimization methods proceed by changing an initial solution to another solution by a variety of operations. A collection of operations on $D$ is elementary if it has the following properties:

(a) If $y \in D$ can be obtained from $x \in D$ by an elementary operation, then $x$ can also be obtained from $y$ by an elementary operation.

(b) Given any two $x, y \in D$ there is a finite sequence of elementary operations which converts $x$ into $y$.

The elementary operations define a connected graph $G$, whose vertices are members of $D$ and whose edges join members of $D$ linked by an elementary operation. Different sets of elementary operations define different graphs. Sometimes, the resolution of an optimization problem is represented as a search in $G$ for an optimal solution.

The neighborhood $N(x)$ of a solution $x \in D$ given by an elementary operation is the set of all vertices adjacent to $x$ in the graph $G$ defined by the operation.

A local optimum is a solution $x \in S$ such that $f(x) \geq f(y)$ for all $y \in N(x) \cap S$.

Let $x^*$ be the optimal solution (or one of the optimal solutions) for the problem. A value $\zeta \in \mathbb{R}$ is an upper bound if $\zeta \geq f(x^*)$; it is a lower bound if $\zeta \leq f(x^*)$.

There are two basic approaches for solving a combinatorial optimization problem: the problem can be solved either exactly – that is, an optimal solution is computed using an exact method or approximately – a heuristic is used to compute an approximate solution. If the problem is NP-hard, then exact methods may demand an exhaustive exploration of the search space. Heuristic strategies, on the other hand, can generate a solution in a feasible amount of time, but in general provide no guarantee of finding the optimal solution.

Next we describe four meta-heuristic methods for solving optimization problems – Greedy heuristics, Hill Climbing, Simulated Annealing, and Genetic Algorithms – and an exact technique, Integer Linear Programming. Hill Climbing, Simulated Annealing and Genetic Algorithms are types of local improvement techniques – they start with a feasible solution for the problem and try to improve it by performing elementary operations.
Greedy Heuristics

Greedy heuristics construct a solution to a problem by following two main rules:

- At each stage of the construction of the solution, the alternatives are analyzed locally and the best choice is taken.
- Previous alternatives are not reconsidered as the heuristic progresses.

An example of a greedy heuristic is Kruskal’s algorithm for computing the minimum spanning tree of a graph [37].

Greedy algorithms are in general quite fast, but they have the drawback of not being able to escape from local minima.

Hill Climbing

Hill Climbing (also called Gradient Descent method) is an iterative algorithm that tries to improve an existing solution by elementary operations. If an operation leads to a better solution, then the current solution is replaced by the new one; otherwise, another operation is tried. This process repeats until no further improvement is possible.

There are a number of variations of the Hill Climbing method. For example, the elementary operations can be based on random changes of the solution or on heuristic strategies. The method can also apply the first operation that causes improvement of a solution, or analyze several operations and execute the one that causes the greatest improvement.

Hill Climbing presents the same advantages and disadvantages of Greedy heuristics.

Simulated Annealing

Simulated Annealing (SA), proposed by Kirkpatrick et al. [107], is a type of Hill Climbing based on the principles of Statistical Mechanics [135]. The method aims to escape from local minima by applying an idea similar to the annealing process, in which liquids are cooled down until assuming a homogeneous form. If the cooling is sufficiently slow, then the molecular structure of the liquid has time to organize itself, resulting in the form of a crystal, which is associated with a state of minimum energy. However, if the annealing is too fast, then the system (liquid) takes an amorphic form that represents a local minimum.
2.1 Basic Concepts

The method simulates the annealing process by starting with a high temperature value \( T = T_0 \) and decreasing \( T \) slowly. For every temperature \( T \), it applies a sequence of movements in order to change the state of the system. New states are obtained in the neighborhood of the current state by random elementary operations. The main rule is that the probability of the system to change from a state with energy \( E_1 \) to a new state with energy \( E_2 \) is given by:

\[
p = \min\{1, e^{\frac{E_2 - E_1}{T}}\}.
\]

This function implies that the system always moves to a new state when \( E_2 < E_1 \). Otherwise, the new state is accepted with probability \( p \). Figure 2.1 shows the Simulated Annealing algorithm. The value of \( K \), the initial temperature and the rate in which \( T \) decreases, among other parameters, have to be chosen carefully.

1. Choose an initial configuration \( \sigma \) for the system and a temperature \( T = T_0 \).
2. Repeat \( K \) times
   (a) Choose a new configuration \( \sigma' \) from the neighborhood of \( \sigma \).
   (b) Let \( E \) and \( E' \) be the energy functions (measuring the costs) for \( \sigma \) and \( \sigma' \), respectively; if \( \text{Random} < e^{(E - E')/T} \) then do \( \sigma \leftarrow \sigma' \).
3. Decrease \( T \).
4. If a stop condition is reached (for example, if \( T \) is too small) then stop; otherwise, go to step 2.

**Figure 2.1:** The Simulated Annealing algorithm.

Constraints can be treated in many ways. One possibility is to consider only solutions that are feasible during the optimization processing. Another approach is to allow infeasible solutions to be generated, but penalize the non-satisfaction of a constraint in the objective function.

Simulated Annealing can be applied to solve optimization problems in general. It is expected to provide very good results, but may demand a considerable amount of runtime.

**Genetic Algorithm**

Genetic algorithms [81] are improvement methods based on Darwin’s Theory of Natural Selection. They are characterized by a cyclic process in which a population of individuals (or chromosomes) evolve. The method consists of three basic steps, which are repeated until a stop condition is reached:
2.1 Basic Concepts

1. Selection of a subpopulation of individuals from the initial population based on their fitness (defined by a fitness function).

2. Execution of genetic operators on the selected individuals in order to generate a population of offsprings. The operators are usually classified as mutations and crossovers. A mutation creates a new individual by copying an existing one and changing part of its structure; a crossover operator produces new individuals by combining parts of two or more existing individuals. The mutation and crossover concepts are related to elementary operations.

3. Replacement of the previous population with the new population.

Genetic algorithms have been applied to many optimization problems, including the TSP [92, 121, 134, 183] and the Bin-packing problem [93, 126, 156, 158]. Their main advantages are the ability to explore several regions of the solution space simultaneously, and the high suitability for parallel and distributed processing. Nevertheless, genetic algorithms also demand much processing time.

Another problem, that is common for Genetic Algorithms and Simulated Annealing, is that these methods in general do not offer a guarantee about the quality of their final solutions.

**Integer Linear Programming**

*Integer Linear Programming* (ILP) is considered the most efficient general technique for solving combinatorial optimization problems in an exact way. The constraints and the objective of an optimization problem are formulated in ILP mathematically as a set of linear functions. A general form of an ILP problem with \( n \) variables and \( m \) constraints is:

\[
\text{Maximize } c_1 x_1 + c_2 x_2 + \ldots + c_n x_n
\]

subject to:
2.1 Basic Concepts

\[ a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \]
\[ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \]
\[ \ldots \]
\[ \ldots \]
\[ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \]

\[ x_1, x_2, \ldots, x_n \geq 0 \]

where \( c_i, a_{ji} \) and \( b_j \) are constants, and \( x_i \) are variables of the problem, for \( i = 1, 2, \ldots, n \) and \( j = 1, 2, \ldots, m \). The formulation above without the integrality constraints (*) characterizes a continuous problem that is referred to only as a Linear Problem (LP). This is usually solved using the simplex method [140].

Two popular methods for ILP are Branch-and-Bound and Branch-and-Cut. Branch-and-bound computes an implicit enumeration tree of LP problems. The tree starts with a single vertex containing a relaxed version of the original ILP problem without the integrality constraints (*). The method executes as follows: for each vertex \( P \) of the tree, which consists of a LP problem, the method analyzes \( P \) by computing its optimal solution \( X \) and verifying whether it is necessary to branch the tree on \( P \). The value of the objective function \( f(X) \) defines a bound for the original ILP problem. The solution \( X \) is checked for a non-integer variable \( x_i, 1 \leq i \leq n \). If all variables in \( X \) are integer, then the ILP problem is already solved. Otherwise, if the bound defined by \( P \) is worse than the best bound found so far, then the vertex \( P \) is removed from the tree. If none of the two previous condition are satisfied, then \( P \) is divided into two subproblems, \( P_1 \) and \( P_2 \). These problems have the same objective function and constraints of \( P \), but include a new bound condition: \( P_1 \) is assigned a constraint \( x_i \leq \lfloor v \rfloor \), and \( P_2 \) is assigned a constraint \( x_i \geq \lfloor v \rfloor + 1 \), where \( v \) is the value of \( x_i \) in the optimal solution for \( P \). The problems \( P_1 \) and \( P_2 \) are then added to the tree as child vertices of \( P \), and the branch process is repeated to another vertex not yet analyzed.

Branch-and-cut is a combination of the Branch-and-bound method with a cutting planes algorithm. The cutting planes technique consists of iteratively adding special constraints to a LP problem in order to get an integer solution. The Branch-and-Cut method tries first to solve a relaxed (LP) version of the ILP problem by using cutting planes. If an integral solution is obtained, then
the problem is considered solved. Otherwise, the algorithm branches as in the Branch-and-Bound method and repeat the same process.

The downside of using the ILP technique is that the optimization problem has to be formulated as a set of linear functions. In many cases, such functions are not adequate to describe the problem. Even when they are adequate, it can be difficult to model the problem as an ILP.

2.1.3 Human-Computer Interaction and Usability Studies

Human-computer interaction has been studied for quite a while, and today helpful guidelines exist for the development of interactive systems. One of the main general aims is to build systems that are “user friendly”. Shneiderman [171] suggests, however, replacing the term “user friendly” by some clearer and more measurable human-factor goals:

1. **Time to learn** – minimizing the amount of time necessary to learn to use the system.

2. **Speed of performance** – reducing the amount of time necessary to perform a task with the system.

3. **Rate of errors by users** – minimizing the rate of errors when performing tasks.

4. **Retention over time** – improving learning, so that users still remember how to use the system after a long time without operating it.

5. **Subjective satisfaction** – increasing user’s satisfaction with the system.

These goals can be achieved by adopting principles that have been successfully applied in many systems:

1. Implementing consistent and compatible ways for the user to enter data into the system and for the system to display data.

2. Allowing frequent users to use shortcuts to speed up their main actions.

3. Reducing the short-term memory load, that is the amount of information that the users have to remember when performing a task with the system. This can be done by adding extra information to the interface in order to help the users.

4. Offering informative feedback for the users’ actions.
5. Preventing the users making serious errors or helping them to correct the problem without having to redo the entire work.

6. Permitting easy reversal of actions.

There are several types of systems regarding interaction styles. Shneiderman [171] discusses some examples of styles: menu selection, form filling, command-language based approaches, natural language and direct manipulation. Here we focus on direct manipulation.

Direct manipulation implements a visual representation of the problem to be solved by the user, so that he or she can perform tasks by directly manipulating objects. The main features of this approach are: (1) a continuous representation of the objects and actions of interest; and (2) the use of physical actions such as clicking on buttons and dragging and dropping objects on the screen, instead of using complex syntax commands. Furthermore, direct manipulation allows incremental reversible operations that have an immediate visible effect on the object of interest. Examples of Direct-Manipulation systems include text editors, electronic spreadsheets, computer-aided design (CAD) systems, games and geographic information systems.

A benefit in using direct manipulation pointed out by Shneiderman is that novice users can learn basic functions quickly by playing with the system, or through a demonstration by a more experienced user. Intermittent and frequent users also benefit from the approach as it makes operational concepts easy to remember and allows fast execution of tasks. Moreover, users receive immediate feedback of their actions, and can change the direction of their activity if these results are not positive.

There are, however, problems with the direct manipulation approach, which are related to the use of a visual representation:

- The representation may be too large and take more than the visible screen space to be properly displayed. Therefore, repetitive and annoying scrolling of the visualization may be necessary.
- The visual representation may be meaningful for the designer of the system, but not to the final user.
- The representation may also be misleading; for example, the user may understand its general meaning, but may incorrectly interpret how to interact with it.
- Direct interaction with the visual representation may not be as efficient as typing a command for some particular problems.
2.1 Basic Concepts

Despite these problems, direct manipulation still appears as a more natural and easy-to-remember interaction style than other approaches such as menu selection and command typing. This is the case for most of the interactive systems discussed in the next sections. Direct manipulation can also be combined with other approaches when good visualizations for some objects of interest cannot be found.

All interactive systems involve tasks that must be performed by the user, as well as tasks that are to be executed by the computer. An important issue for the designer of such systems is to find a good balance between automation and human control. Shneiderman recommends simplifying the user’s role by eliminating human action when no judgment is required, and avoiding repetitive, tedious, and error-prone tasks. Instead, the user should concentrate on creative tasks, critical decisions, strategic planning, and on coping with unexpected situations. The computer, on the other hand, should be used to manipulate large volumes of data, to execute repetitive preprogrammed action reliably, to monitor and control well pre-specified events, and to execute complex mathematical and logical operations.

The design of an interactive system must be tested to verify whether it attends the human-factor goals specified previously. The system can be tested not only after implementation, but also in much earlier stages of its development in order to aid with the choice and validation of design solutions. Tichy [184] comments that many Computer Science papers (including publications in Software Engineering) have unsupported claims; he proposes that computer scientists should do more experimentation.

Several approaches exist for evaluating a system; examples are pilot studies of design solutions and rapid prototyping. System evaluation can be done through interviews and group discussions, surveys, and controlled human experiments.

Evaluation based on human experiments follows methodologies from psychology. The basic steps for this type of experiments involve stating a testable hypothesis, building a well controlled setup, measuring the aspects of interest with a significant number of subjects, and analyzing the results using statistics.

Sometimes large human experiments are not possible because the application depends on expert domain knowledge, and expert users are unwilling or not available in a sufficient number. Nielsen [144] suggests an evaluation technique for this situation called *heuristic evaluation*.

Heuristic evaluation is a systematic inspection of a user interface design for usability problems. The evaluation is performed by a small number of evaluators who examine the interface and try
to identify good and bad aspects of it, according to a list of recognized usability principles. Each individual evaluator inspects the interface separately; only after that are they allowed to communicate and aggregate their findings. The best number of evaluators depends on the application, but it has been recommended to use about five and at least three persons. An individual evaluation section usually takes between one to two hours. The results of the examinations can be recorded in written reports by each evaluator, or by an observer who is present in the sessions and annotates the comments vocalized by the evaluators. The observer can also assist with the operation of the system (this may be necessary when the system is not fully implemented, or when the interface is too complex and the evaluator has no time to be trained to use it). Furthermore, the observer must answer evaluators’ questions about the functioning of the system.

Some studies have shown that domain experts identify more usability problems than novice users. Moreover, evaluators that are not only domain experts but are also experienced with the evaluation of systems have provided much better feedback.

### 2.1.4 Information Visualization

Information Visualization is an emerging area that studies ways of amplifying cognition of ideas, processes, and phenomena through visual representations. Techniques from this area can be used either to communicate information or to create and discover it.

There are many techniques for Information Visualization and we do not intend to cover them in this thesis. For a good survey of the field, we recommend the books of Card et al. [31] and Spence [174]. We only illustrate a few approaches in order to show how broad and fascinating this area is. Most information visualization techniques explore visual aspects such as dimensionality, color, object size and shape, and animation or dynamic changes of visual representation. Some ideas involve concepts of direct manipulation, where users can rotate, move or select objects on the screen in order to gain a better understanding of a phenomenon or concept.

A typical example of dimensionality is a 3D visualization of a mathematical function based on two variables (see Figure 2.2). This visualization supports understanding of mathematical concepts such as continuity and minimum value. The user may also change the view point in order to see regions of the graphics that are currently occluded.

Color and other graphical properties such as size and shape can be employed to classify objects and to emphasize information that requires immediate attention of the user. Figure 2.3 shows an example where color is used to discriminate regions and periods of the year where bush fires are