WORKING PAPER

ITLS-WP-16-23

Introducing pattern graph rewriting in novel spatial aggregation procedures for a class of traffic assignment models

By
Mark P.H. Raadsen[1], Michiel C.J. Bliemer[1] and Michael G.H. Bell[1]

[1] Institute of Transport and Logistics Studies (ITLS), The University of Sydney Business School, Sydney, Australia

December 2016

ISSN 1832-570X

INSTITUTE of TRANSPORT and LOGISTICS STUDIES

The Australian Key Centre in Transport and Logistics Management

The University of Sydney

Established under the Australian Research Council's Key Centre Program.

NUMBER: Working Paper ITLS-WP-16-23

TITLE: **Introducing pattern graph rewriting in novel spatial aggregation procedures for a class of traffic assignment models**

ABSTRACT:

In this study two novel spatial aggregation methods are presented compatible with a class of traffic assignment models. Both methods are formalized using a category theoretical approach. While this type of formalization is new to the field of transport, it is well known in other fields that require tools to allow for reasoning on complex structures. The method presented stems from a method originally developed to deal with quantum physical processes. The first benefit of adopting this formalization technique is that it provides an intuitive graphical representation while having a rigorous mathematical underpinning. Secondly, it bears close resemblances to regular expressions and functional programming techniques giving insights in how to potentially construct solvers (i.e. algorithms). The aggregation methods proposed in this paper are compatible with traffic assignment procedures utilising a path travel time function consisting out of two components, namely (i) a flow invariant component representing free flow travel time, and (ii) a flow dependent component representing queuing delays. By exploiting the fact that, in practice, most large scale networks only have a small portion of the network exhibiting queuing delays, this method aims at decomposing the network into a constant free flowing part to compute once and a, much smaller, demand varying delay part that requires recomputation across demand scenarios. It is demonstrated that under certain conditions this procedure is lossless. On top of the decomposition method, a path set reduction method is proposed. This method reduces the path set to the minimal path set which further decreases computational cost. A large scale case study is presented to demonstrate the proposed methods can reduce computation times to less than 5% of the original without loss of accuracy.

KEYWORDS: ***Traffic Assignment; Spatial aggregation; Category theory; Graph rewriting***

AUTHORS: **Raadsen, Bliemer and Bell**

Acknowledgement:

We would like to acknowledge the valuable comments we received from Aleks Kissinger which greatly helped develop the category theoretical underpinning of this work. This research is partly funded by the Australian Research Council Linkage Project LP130101048.

CONTACT:

INSTITUTE OF TRANSPORT AND LOGISTICS STUDIES (H73)

The Australian Key Centre in Transport and Logistics Management

The University of Sydney NSW 2006 Australia

Telephone: +612 9114 1813

E-mail: business.itlsinfo@sydney.edu.au

Internet: http://sydney.edu.au/business/itls

DATE: December 2016

1. Introduction

Aggregation methods in transport have been studied since the late ‘60s of the 20th century. Initially the main objective was to reduce computational cost as a result of the limited amount of computing resources available for the task at hand. Today governments and industry are shifting more and more towards operating multiple transport models alongside each other. In such environments (dis)aggregation plays an increasingly important role, not only to reduce computation times, but also in providing conversions between different levels of granularity. In doing so, one mainly aims to remove/introduce redundant/relevant detail. However, it should be noted that it depends on the application at hand what detail can be considered redundant/relevant.

Given a more recent emphasis on consistency and the observation that an aggregation method is only as effective as the application context it was designed for, it is surprising to see that very little research regarding aggregation methods take these two components explicitly into account.

In this research the application context considered is one where the transport network (supply) is assumed fixed, while demand can vary per scenario $s \in S$. There is a multitude of applications that follow this protocol and could benefit from the proposed method. One can think of matrix calibration/estimation procedures, quick scan methods and/or demand scenario comparisons.

Most aggregation methods in the literature are described either informally or in terms of an algorithm. This can make it difficult to analyze what exactly is being proposed. To address this, we formalize our methodology by adopting and modifying a method that is well known in the field of quantum processing (Coecke and Duncan, 2011) and (linear) logical reasoning, but has so far not been used in the field of transport. The proposed adaptation is grounded in category theory and it is considered attractive in this context since it provides an intuitive graphical interpretation, while at the same time has a rigorous mathematical underpinning.

The concept of delay is important in this paper. We distinguish between two types of delay: Hypercritical delay and hypocritical delay (Cascetta, 2009). Hypercritical delay is the

queuing delay when travel demand exceeds infrastructure supply. This delay is related to the congested part of the fundamental diagram where flow decreases with increasing density. Hypocritical delay on the other hand is the delay one experiences when there are no queues, but drivers can no longer traverse a road segment at free speed due to interactions with other vehicles. This delay is related to the uncongested part of the fundamental diagram where flow increases with increasing density. Therefore, the total travel time on a link or path consists of free flow travel time plus hypocritical delay plus hypercritical delay.

On the part of the aggregation methodology, we make a number of simplifying assumptions are to support a (potential) lossless outcome for our class of compatible traffic assignment models: (i) Hypocritical delay is assumed to be zero. This is an often made assumption in which only free flow travel time and hypercritical (queuing) delays are considered, and is consistent with for example the well-known triangular fundamental diagram (Newell, 1993) and the bottleneck model (Vickrey, 1969). (ii) The traffic assignment procedure is path based with a given set of paths.

For simplicity we assume that path costs consist solely of travel time, although this assumption can be relaxed. The path travel cost function compatible with the defined class of models is then given by (1).

$$c_p^{(s)} = \tau_p^I + \underbrace{\tau_p^{II(s)}(f_p^{(s)}) - \tau_p^I}_{\text{hypocritical delay}} + \underbrace{\tau_p^{III(s)}(f_p^{(s)})}_{\text{hypercritical delay}} \approx \tau_p^I + \tau_p^{III(s)}(f_p^{(s)}), \tag{1}$$

where $f_p^{(s)}$ denotes the (desired) path flow for a path $p \in P$ under demand scenario $s \in S$. Here $\tau_p^{II(s)}(f_p^{(s)})$ represents the flow dependent uncongested path travel time (including hypocritical delay), and $\tau_p^{III(s)}(f_p^{(s)})$ represents the flow dependent congested travel time (hypercritical delay). Uncongested travel time $\tau_p^{II(s)}(f_p^{(s)})$ can be decomposed into flow independent minimum free flow path travel time τ_p^I (based on maximum link speeds), and hypocritical delay $\tau_p^{II(s)}(f_p^{(s)}) - \tau_p^I$, where the latter is assumed to be zero in this context. As a result, the path travel time in Equation (1) can then be decomposed into a part that is flow independent and a part that is flow dependent only in case of congestion. The efficiency of

the proposed aggregation concept hinges on exploiting the possibility of decomposing path travel times and utilise it to decompose the entire transport network.

This results in a free flow network and a (queuing) delay network. Since the free flow network is independent of flow, τ_p^I only needs to be computed once across all demand scenarios. The delay network on the other hand requires computation for each demand scenario separately. However, since the portion of the network experiencing queuing delays is typically small, a significant reduction in computation time is realised. Path travel times are reconstructed by combining the free flow travel time and delay travel time of the two networks.

The proposed aggregation method is especially effective in a context where queues are assumed vertical, i.e. do not spillback. However, one can also apply it in a (dynamic) context with horizontal queues. Finally, note that one can even adopt this approach when violating assumption (i); an example being traditional static assignment with a BPR function (Bureau of Public roads, 1964). In such cases $\tau_p^{II(s)}(f_p^{(s)}) - \tau_p^I \geq 0$ and the methodology becomes lossy rather than lossless. It is therefore not explicitly considered in the remainder of this work.

1.1. Contributions and outline

This work makes the following contributions. First, we introduce a category theoretical approach in formalizing transport network transformations by adopting and adapting pattern graph rewriting techniques. To the best of the author's knowledge this is the first time such a method is used to formalize methodology in the field of transport. Secondly, we propose a novel network aggregation procedure based on path travel time decomposition as well as a novel zonal aggregation procedure that can be applied on top of the decomposition method utilising a path aggregation scheme. Both methods are lossless under some (discussed) conditions. Thirdly, we demonstrate applicability via theoretical examples and a real world large scale case study adopting a recently developed quasi-dynamic traffic assignment model with residual point queues. Preliminary results show potential for computational reductions to less than 5% of the original cost, depending on the demand scenario.

This paper is organized as follows. Section 2 discusses the current state of spatial aggregation procedures in traffic assignment. Section 3 introduces string graphs and how they can represent transport networks. Section 4 provides an introduction into category theory and pushouts and introduces pattern graphs. Section 5 formalizes graph rewriting with rewrite patterns which are used to describe the two aggregation methods discussed in Sections 6 and 7, respectively. Section 8 presents two case studies, one on a small theoretical network and one on a large real network. Conclusions and future research are outlined in Section 9.

2. Spatial aggregation in the literature

Two types of spatial aggregation are typically distinguished in the transport literature: network aggregation which impacts on the road infrastructure, i.e. nodes and links, and zonal aggregation which aggregates the zones (and virtual links connecting the zones to the network).

2.1. Network aggregation

One can differentiate between network extraction and network abstraction methods within the network aggregation paradigm (Chan et al., 1968; Connors and Watling, 2008). Network extraction directly removes nodes and links from the network. Examples of this type of procedure can be found in Chang et al. (2002), Long and Stover (1967) and Bovy and Jansen (1983). Network abstraction on the other hand, replaces nodes and links with something else, in order to mimic the original situation. Chen et al. (1968) argue that network extraction is an undesirable approach because it reduces capacity on the network, unrealistically diverts traffic and breaks network connectivity. Based on these findings Chan (1976) proposes a network abstraction method using zonal bypasses adopting a traditional static network loading procedure.

Network decomposition can be regarded as an alternative to network aggregation, instead of aggregating the network it decomposes the network in order to simplify the problem. Examples of this approach, proposing a traffic transfer decomposition technique, can be found in Barton and Hearn (1979) and Hearn (1984), where they determine a deterministic user equilibrium by solving a mathematical programming problem. For a comprehensive literature review on network topology decomposition (mostly related to queuing theory) we refer to Osorio and Bierlaire (2009). All aforementioned methods only consider static capacity restrained traffic assignment (i.e., no capacity constraints and residual queues) and do not take the application context into account.

2.2. Zonal aggregation

In an urban planning or demand modelling context, zonal aggregation is often associated with the procedure on how to construct zones. This is also known as zoning effects (Openshaw and Taylor, 1979; Paez and Scott 2004). However, this research considers a traffic assignment context, as such the original zoning structure is assumed given (zones based on a detailed level of postal codes for example) and we only look at the scaling effect of zoning. This entails grouping of existing zones. This type of zonal aggregation is termed the "spatial aggregation problem" by Daganzo (1980a). Daganzo solved the problem by embedding a zonal aggregation procedure into the Frank and Wolfe (1956) algorithm. A generalised version extending this to a continuum approach (Newell, 1979) also exists (Daganzo, 1980b).

Some studies combined a zonal aggregation and network aggregation procedure to create a more comprehensive approach (Jeon et al., 2010; Bovy and Jansen, 1983). They remove categories of links deemed not important enough. As a second step the zones, residing within the "holes" formed by the network aggregation procedure, are grouped. Chang et al. (2002) adopt a similar approach, only they group zones according to functional groups. The main problem of these methods is twofold, namely (i) they only consider the traditional static traffic assignment approach and more importantly, no justification is provided on why removing certain links, or grouping certain centroids is reasonable, and (ii) all combined

network and zonal aggregation approaches rely on network extraction which is an undesirable approach if one can avoid it.

As stated the proposed aggregation method addresses aforementioned issues by explicitly stating the application context the method is designed for. Furthermore, the procedure itself is formalised in category theory. Since this is the first time such a method is applied in the field of transport some basic concepts are explained while introducing the necessary notation.

3. Graph representation in a category theoretical context

Category theory comes with entirely new terminology that will be unfamiliar to most transport scientists. This section as well as Sections 4 and 5 lean heavily on definitions originally introduced in Kissinger et al. (2014). All Definitions adopted from Kissinger (2014), but adapted to suit the specific needs of the aggregation method are indicated with an asterisk (*). To correctly construct the algebra these definitions require revisiting in this paper, especially since many of them have been altered. We urge the reader not to worry if some of the more mathematical constructs, especially in Sections 5 and 6, are not immediately understood. One does not necessarily need to understand every intricate detail of how this algebra is constructed in order to take advantage of the benefits of the final result, much the same as not every driver understands how the engine of its car works. Applying the graphical rewriting method in the presented aggregation methodology, rather than the math underpinning this algebra is discussed from Section 6 and onward.

Category theory revolves around the notion of objects and arrows. Objects are instances of a particular category while arrows, named morphisms, typically (but not necessarily) represent operations on objects that allow for transformations from one object to another. In this section we will define several categories of graphs used to represent our transport network and paths that go through this network. Let start with how we construct the category of graph from the category of sets. A graph $\mathcal{G}$ contains edges $e \in E_{\mathcal{G}}$ and vertices $v \in V_{\mathcal{G}}$ and the two can be linked via (2)[1].

[1] Diagrams such as the one depicted in (2) are valid category theoretical constructs and have a closer resemblance to an

$$E \underset{trg}{\overset{src}{\rightrightarrows}} V \tag{2}$$

Both *src* and *trg* are functions yielding the source (upstream vertex) or target (downstream vertex) of the edge respectively, observe that this implies that each edge and vertex is both uniquely labelled as well as directed. In addition we define that when $src(e) = v$, edge e is an out-edge of v. Conversely, it is an in-edge when $trg(e) = v$.

The definition of the category of graphs from the category of sets can be achieved via a functor category, which roughly speaking, imposes the additional rules stated in (2) onto the category of sets to construct the category of graphs, or more formally:

Definition 3.1[*]: **Graph** is the category of directed labelled g*raphs*, which is a functor category denoted as $\mathbf{Set}^G$ where **Set** is the category of sets and G represents (2), i.e. two morphisms between two objects of the category of sets

Let us now define the successors of edge v through $succ(v)$. This is the set of all vertices v' that have an in-edge e which is an out-edge of v. Similarly, the set of vertices v'' which have an out-edge e which is an in-edge of v is known as the predecessors of v, i.e. $pred(v)$.

In transport networks, intersections (nodes) are often represented by vertices while road segments (links) are indicated by edges (Figure 1a). However, for this research an alternative representation is adopted. The reason for this is that it will simplify the final graph rewriting method as it obviates the need for explicit node representation.

equation than a figure. Therefore they are treated as equations rather than figures in the remainder of this paper.

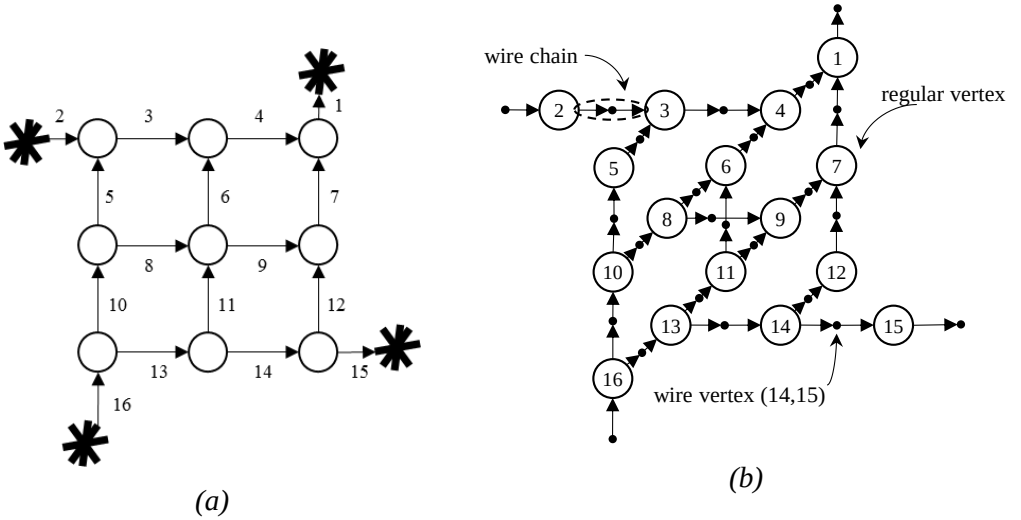
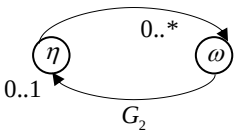


Fig. 1. (a) Common representation of example network of a Manhattan type network with four travel zones. (b) String graph representation of the same network shown in (a).

Links are modelled as vertices while nodes are indirectly represented through turns, where each turn is modelled through a so called wire chain. Each wire chain consists of two edges and a single special vertex termed wire vertex, see Figure 1(b). This network representation can be categorised as a string graph (Dixon et al. 2010) which is the discrete version of a string diagram. These diagrams were originally used to describe calculations on tensor networks (Joyal and Street, 1991). One of the benefits of representing a transport network as a (string) graph is that it allows to apply specific graph rewriting rules using pushouts, inspired by recent work from Kissinger et al. (2014). These (recursive) rewriting rules are used to formalise the presented aggregation method. To define a category of graphs that has two different types of vertices one can define a type graph. A type graph can be used to formalise the allowed interaction between the differently typed components of the original graph category. It is a way to make a category more specific. For example the type graph that defines the category of string graphs as shown in Figure 1(b) is given by G_2:



(3)

Wire vertices are represented through type ω and "regular" vertices through type η, i.e. $V = \eta \cup \omega$. Multiplicities (e.g. 0..1, 0..*) impose the lower and upper bound on edge cardinality, identical to multiplicities used in class diagrams in the field of computer science. This means that each wire vertex $v \in \omega$ is allowed to connect towards at most one regular vertex $v' \in \eta$, while each regular vertex can have zero or more (a '*' represents infinite) connections to wire vertices.

Definition 3.2* SSGraph: The category **SSGraph** of *simplified string graphs*[2] is the full subcategory of the slice category Graph/G_2 adhering to the multiplicities indicated. Furthermore, only a single wire vertex is allowed to exist in between two regular vertices on any string graph $\mathcal{G}$, such that each wire vertex can be uniquely defined through its adjacent regular vertices, see Figure 1(b) for an example.

While simplified string graphs can represent our transport network, each path can be defined as a string graph as well, albeit a slightly different one. Each path $p \in P$ has a representing linear string graph (Definition 3.3) denoted by $\mathcal{P}_p \subseteq \mathcal{G}$, where $\mathcal{G}$ is a simplified string graph representing the entire transport network.

Definition 3.3 LSGraph: The category **LSGraph** of *linear string graphs* is identical to **SSGraph** except for its multiplicities. **LSGraph** allows only a single wire to enter and a single wire to exit a regular vertex, hence the name *linear string graph*.

Both simplified string graphs and linear string graphs have a boundary. In both cases the boundary is defined as the union of locations in the graph where a wire vertex lacks an ingoing or outgoing edge, see also Definition 3.4 or Figure 2 for an example. This notion of boundaries will become important in the following section when defining our algebra.

[2] This category is termed 'simplified' because the typical definition of a string graph does allow for wire chains containing multiple wire vertices, while we limit ourselves to a single wire vertex to suit our needs.

Definition 3.4* Boundary: The boundary of a (linear) string graph is defined in terms of its wire vertices. Given a string graph $\mathcal{G}$ the set of inputs $\text{In}(\mathcal{G}) \subseteq \omega_{\mathcal{G}}$ is determined such that when $v \in \text{In}(\mathcal{G})$, it holds that $pred(v) \cap \eta_{\mathcal{G}} = \varnothing$. Similarly, $\text{Out}(\mathcal{G})$ is determined such that when $v' \in \text{Out}(\mathcal{G})$, $succ(v') \cap \eta_{\mathcal{G}} = \varnothing$. The intersection with $\eta_{\mathcal{G}}$ ensures an unaltered interpretation of what constitutes a boundary in case additional vertex types are introduced. The boundary is given by $\text{Bound}(\mathcal{G}) = \text{In}(\mathcal{G}) \cup \text{Out}(\mathcal{G})$.

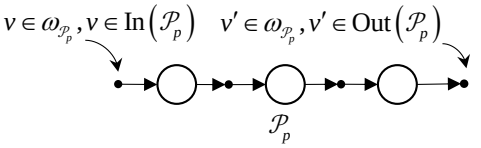


Fig. 2. Example linear string graph $\mathcal{P}_p$ for some path p and its boundary.

4. Graph transformation via pushouts

The graph categories defined in Section 3 are in fact sets with some additional structure imposed by (2) and (3). If one performs a mapping on an object of the category of graphs and the result of this function is again a graph of the same category, the mapping is said to be structure preserving. Such as a structure preserving mapping is known as a (homo)morphism in category theory. Given we only consider (sub)categories of graphs and sets in this paper one can think of morphisms as a generalisation of an ordinary function that acts upon sets (although this does not hold in general). For example $m: \mathcal{G} \to \mathcal{G}'$ denotes morphism m that maps object $\mathcal{G}$ to $\mathcal{G}'$, where the former is known as the domain and the latter as the codomain of m. A composition of multiple morphisms is typically depicted in a diagram by combining their representative arrows.

To provide some context for the reader unfamiliar with category theoretical constructs Table 1 outlines some of the more common morphisms and their functional equivalents, i.e. interpretations in the context of the category of sets/graphs as considered in this paper.

Table 1. Morphisms and their (better known) functional equivalents for the category of sets/graphs.

Morphism type	Homomorphism synonym	Set/graph functional equivalent	Interpretation
Monomorphism	Injective homomorphism	Injective function	Each element in the domain is mapped to maximum one element in the codomain
Epimorphism	Surjective homomorphism	Surjective function	Each element in the codomain was mapped from minimum one element in the domain
Isomorphism	Bijective homomorphism	Bijective function	Every element in domain, codomain has a single mapped element in codomain, domain respectively.
Identity morphism	-	Identity mapping	An object is mapped to itself

One example of morphism composition is the pushout. While this is a known method in many areas of research it is, as far as the authors are aware, new to the field of transport. An introduction into the basic concept is therefore warranted. Pushout graph rewriting was first proposed by Ehrig et al. (1973). Popular constructs are the single pushout (SPO) and double pushout (DPO). In this paper we use them to modify the transport network and path representations.

Definition 4.1 Single Pushout (SPO): Let $m_1 : \mathcal{K} \to \mathcal{G}'$ and $m_2 : \mathcal{K} \to \mathcal{G}''$ be morphisms, the pushout of m_1 and m_2 is the 'glued' object $\mathcal{G}' \cup \mathcal{G}''$ with morphisms $m_1' : \mathcal{G}' \to \mathcal{G}' \cup \mathcal{G}''$ and $m_{2'} : \mathcal{G}'' \to \mathcal{G}' \cup \mathcal{G}''$ such that the diagram in Figure 3 commutes[3].

[3] Given multiple paths exist from one object to another, the diagram commutes when the result of the two paths is identical.

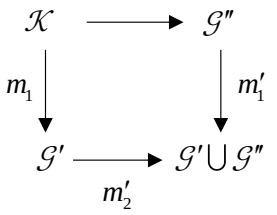


Fig. 3. Pushout diagram, based on Ehrig et al. (1973).

The objective of this pushout is to glue objects $\mathcal{G}'$ and $\mathcal{G}''$ together rather than obtaining the disjoint combination of the two. To achieve this, object $\mathcal{K}$ serves as a common interface. The effect of this interface is that all nodes and links that can be 'glued' are identified by $\mathcal{K}$, which in the context of graphs can be interpreted as $\mathcal{G}' \bigcap \mathcal{G}''$. Due to the fact that this pushout commutes, the minimal 'glued' result graph $\mathcal{G}' \bigcup \mathcal{G}''$ is obtained.

Pushouts can be used to define (graph) rewriting rules. Löwe (1993) showed that rewriting rules in DPO format are in fact special cases of SPO. Since SPO is a simpler construct, the reader is informally introduced to the intuition of this concept using an SPO example. We delay formal definitions until Section 5.

To apply a rewrite rule, one first identifies a subgraph $\mathcal{L}$ of some reference graph $\mathcal{G}$, this is called a *matching* and denoted by injective homomorphism $\mu : \mathcal{L} \to \mathcal{G}$. $\mathcal{L}$ represents the left hand side of rewrite rule $\rho : \mathcal{L} \to \mathcal{R}$, with $\mathcal{R}$ representing the right hand side after the rewrite. Since the SPO commutes, the changes applied via morphism ρ are also applied to $\mathcal{G}$ such that it yields rewritten graph $\mathcal{H}$, which is identical to graph $\mathcal{G}$ except that $\mathcal{L}$ has been replaced by $\mathcal{R}$. Figure 4(a) illustrates the SPO and Figure 4(b) shows a concrete example.

Any dangling vertices that result from this transformation are deleted. In case a node or link in $\mathcal{R}$ does not exist in $\mathcal{L}$ it is added to $\mathcal{H}$. For a comprehensive introduction into category theory and pushouts see for example Awodey (2010) or Simmons (2011).

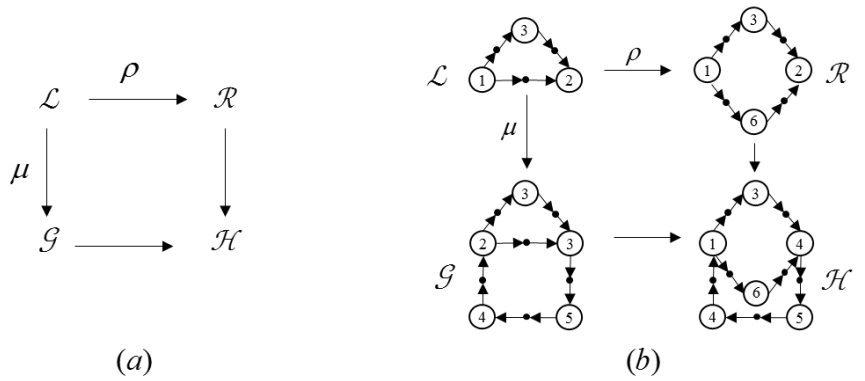
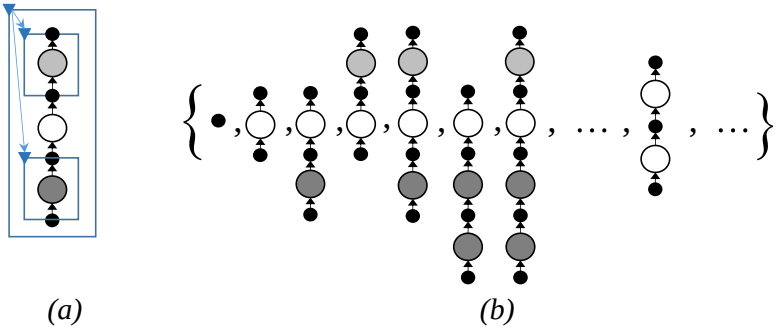


Fig. 4. (a) SPO Rewrite rule notation (b) concrete SPO rewrite rule example for an instance of the category simplified string graph.

4.1. Graph families and pattern graphs

To modify graphs, i.e. our transport network, instances of rewrite rules are to be defined. As we will see later, the rewrite rules required for our aggregation method exhibit a recurring pattern. This results in a large, if not infinite, number of rewrite rules because each possible rule has to be defined explicitly. Dixon and Duncan (2009) first introduced the concept of !-boxes (‘bang’ boxes) to circumvent this problem. This idea was further extended by Dixon and Kissinger (2013) and Kissinger et al. (2014). It allows the specification of an infinite family of graphs in a single visual representation. As an example Figure 5 depicts a !-box around the top (light grey) vertex, which means this vertex can be repeated zero or more times. Similarly, there is a !-box containing the bottom (dark grey) vertex. Additionally, there is a !-box around all three vertices such that any pattern of the three vertices can be repeated zero or more times, with a minimum of a single white vertex in each repetition.



(a) *(b)*

Fig. 5. (a) Example sequential !-box pattern graph and (b) the infinite family of concrete instances of linear string graphs that it represents.

This expression that represents an infinite family of graphs is underpinned by defining operations on (sequential) !-boxes. All operations can be repeated zero or more times and (sequential) !-boxes can also be nested. One can think of this method as the equivalent of pattern matching in regular expressions or the functional programming paradigm from computer science. Due to this close resemblance, this technique has the additional benefit of providing direct insights regarding the construction of solution algorithms. While the original !-boxes in Kissinger at al. (2014) used parallel inspired operations, in the context of this study, three sequential operations are proposed, namely a sequential copy, sequential kill and drop operation (Figure 6).

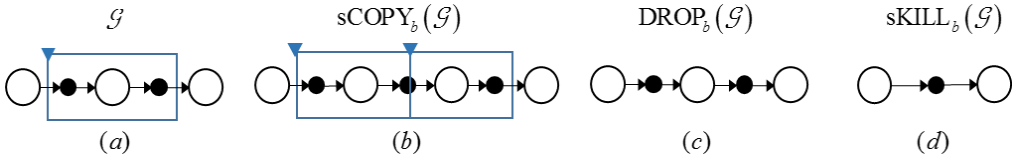


Fig. 6. (a) Example linear string graph $\mathcal{G}$, with a single (sequential) !-box and the result of applying a sequential copy operation (b) on $\mathcal{G}$, a drop operation (c) on $\mathcal{G}$ or a sequential kill operation (d) on $\mathcal{G}$.

To formalize these operations a new type of vertex is required: The s!-vertex (‘sequential bang’ vertex). This requires a new type graph G_3 as depicted in (4) which in turn can be used to create a more specific category including s!-vertices. Each s!-vertex on a pattern

graph $\mathcal{G}$ is denoted by $b \in !_{\mathcal{G}}$. Figure 7(a) shows the proposed schematic box notation while Figure 7(b) depicts the equivalent traditional version of the same graph.

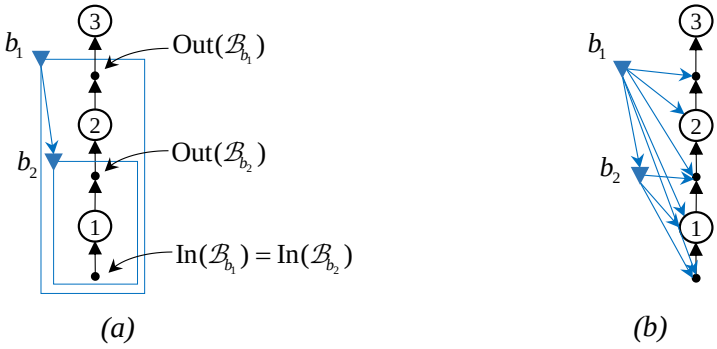
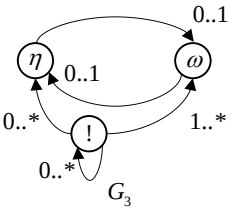


Fig. 7.*: (a) Schematic depiction of s!-boxes and their boundaries, (b) traditional depiction via explicit edges.

Each s!-box is denoted by $\mathcal{B}_b$, with $b \in !_{\mathcal{G}}$. $\mathcal{B}_b$ represents the full subgraph where its vertices are the successors to *b*, i.e. *succ*(*b*). Each $\mathcal{B}_b$, is required to have a single wire vertex that represents its incoming boundary and a single wire vertex that represents its outgoing boundary, i.e. $|\mathrm{In}(\mathcal{B}_b)| = |\mathrm{Out}(\mathcal{B}_b)| = 1$. This ensures that the boundary is invariant under the supported operations.



(4)

Definition 4.2* LSPatGraph: A graph $\mathcal{G}$ which is an instance of the full subcategory of **Graph**/G_3 is known

as a *(linear sequential) pattern graph* (**LSPatGraph**) when:

- The full subgraph with $\eta_{\mathcal{G}} \cup \omega_{\mathcal{G}}$ is a linear string graph,

- The full subgraph $!_{\mathcal{G}}$ is partially ordered[4].
- s!-box $\mathcal{B}_b$ does not overlap with another bang box $\mathcal{B}_{b'}, b \neq b'$ on its internal structure $\mathcal{B}_b \setminus \text{Bound}(\mathcal{B}_b)$ unless $\mathcal{B}_{b'}$ is fully contained in $\mathcal{B}_b$, i.e. if $\left((\mathcal{B}_b \setminus \text{Bound}(\mathcal{B}_b)) \cap (\mathcal{B}_{b'} \setminus \text{Bound}(\mathcal{B}_{b'}))\right) \neq \varnothing$, with then $\mathcal{B}_b \subseteq \mathcal{B}_{b'}$. The latter case is termed *nesting*.

The conditions in Definition 4.2 ensure that the desired operations and rewrite rules work as expected. The three required operations are formalized as follows:

Definition 4.3 $\text{sCOPY}_b(\mathcal{G})$: For pattern graph $\mathcal{G}$ and s!-box $\mathcal{B}_b, b \in !_{\mathcal{G}}$ the sequential copy is denoted by $\text{sCOPY}_b(\mathcal{G})$. Define $\mathcal{B}_{bb}$ as the inclusion pushout of $\mathcal{B}_b$ with itself, where the first copy of $\text{Out}(\mathcal{B}_b)$ is identified with the second copy of $\text{In}(\mathcal{B}_b)$, indicated below through wire vertex $i \in \omega_{\mathcal{B}_b}$:

$$\begin{array}{ccc} i & \longrightarrow & \mathcal{B}_b : \text{In}(\mathcal{B}_b) = i \\ \downarrow & & \downarrow \\ \mathcal{B}_b : \text{Out}(\mathcal{B}_b) = i & \longrightarrow & \mathcal{B}_{bb} \end{array} \tag{5}$$

Then $\text{sCOPY}_b(\mathcal{G})$ is defined through the following DPO, with $\mathcal{G} \setminus_g \mathcal{B}_b$ being the pushout complement (see also section 5.1):

$$\begin{array}{ccccc} \mathcal{B}_b & \longleftarrow & \text{Bound}(\mathcal{B}_b) & \longrightarrow & \mathcal{B}_{bb} \\ g \downarrow & & \downarrow & & \downarrow \\ \mathcal{G} & \longleftarrow & \mathcal{G} \setminus_g \mathcal{B}_b & \longrightarrow & \text{sCOPY}_b(\mathcal{G}) \end{array} \tag{6}$$

Definition 4.4[*] $\text{DROP}_b(\mathcal{G})$: $\mathcal{G} \setminus b$.

[4] This ensures that nested s!-boxes can be properly supported in rewrite rules.

Definition 4.5 $\text{sKILL}_b(\mathcal{G})$**:** Sequential kill is the intuitive inverse of $\text{sCOPY}_b(\mathcal{G})$. A DPO transforms s!-box $\mathcal{B}_b$ to single wire vertex i, which is identified with both $\text{In}(\mathcal{B}_b)$ and $\text{Out}(\mathcal{B}_b)$, i.e. they are merged into the same vertex and the bang box itself is no longer included:

$$\begin{array}{ccccc} \mathcal{B}_b & \longleftarrow & \text{Bound}(\mathcal{B}_b) & \longrightarrow & i \\ {\scriptstyle g}\downarrow & & \downarrow & & \downarrow \\ \mathcal{G} & \longleftarrow & \mathcal{G} \setminus_g \mathcal{B}_b & \longrightarrow & \text{sKILL}_b(\mathcal{G}) \end{array} \tag{7}$$

To be able to construct concrete instances of linear string graphs as depicted in Figure 5(b) from linear sequential pattern graphs as depicted in Figure 5(a), a formal conversion from the latter category to the former is required. This conversion is supported through Definition 4.6.

Definition 4.6* instantiation: For pattern graphs $\mathcal{G}, \mathcal{H}$, we let $\mathcal{G} \succeq \mathcal{H}$, i.e. $\mathcal{H}$ succeeds $\mathcal{G}$ if and only if $\mathcal{H}$ can be obtained from $\mathcal{G}$ by applying operations in Definitions 4.2-4.5 zero or more times. When $!_{\mathcal{H}} = \varnothing$, then $\mathcal{H}$ is called an *instance* of $\mathcal{G}$ while the sequence of operations used to obtain $\mathcal{H}$ from $\mathcal{G}$ is called the *instantiation*.

In other words, each instantiation of a linear sequential pattern graph can be considered to be a linear string graph, since it no longer contains any s!-vertices.

5. Rewrite rules with sequential pattern graphs

We now have the tools in place to extend the conventional rewrite rules by embedding pattern graphs into them, allowing for the definition of recursive rewrite rules needed for our aggregation method presented in Section 6 and onward. The first step in achieving this

is to formalize the concept of *linear string graph matching*, which was introduced informally in Section 4.

Definition 5.1* linear string graph matching: A *linear string graph matching* is defined as monomorphism (injective homomorphism), $\mu: \mathcal{L} \to \mathcal{G}$, $\mathcal{L} \subseteq \mathcal{G}$, with $\mathcal{G}$ being a linear string graph. Then $\mathcal{L}$ matches $\mathcal{G}$ via μ.

Observe that $\mathcal{L}$ can now also be an instantiation of a linear sequential pattern graph, which is paramount in formalizing the pattern graph matching concept.

Definition 5.2* pattern graph matching: Given linear string graph $\mathcal{G}$ and sequential pattern graph $\mathcal{L}_!$; if there is an instance of $\mathcal{L}_!$ with instantiation $\mathcal{S}$, that matches $\mathcal{G}$ via μ, then $\mathcal{L}_!$ matches $\mathcal{G}$ via μ under instantiation $\mathcal{S}$.

5.1. Pattern graph rewrite rules

The purpose of matchings from pattern graphs to string graphs is to utilise them in defining rewrite rules.

Definition 5.3* rewrite rule: A span[5] of linear string graphs $\mathcal{L} \xleftarrow{i_1} \mathcal{I} \xrightarrow{i_2} \mathcal{R}$ is called a rewrite rule if: (i) $\mathcal{I}$ is a point graph and (ii) when one restricts morphisms i_1, i_2 to surjections, they result in $i_1: I \to \text{Bound}(\mathcal{L})$ and $i_2: I \to \text{Bound}(\mathcal{R})$ such that $\text{In}_{\mathcal{L}} \to \text{In}_{\mathcal{R}}, \text{Out}_{\mathcal{L}} \to \text{Out}_{\mathcal{R}}$ are isomorphisms.

Definition 5.3 reflects that separate inputs and outputs can be merged or conversely, allow combined input and outputs to be separated as long as this has no impact on the in and outgoing boundary. This means that regardless what graph this rule is matched to, its boundary won't be impacted by the rewrite rule.

[5] A span consist of three objects and two morphisms, where two of the objects share the same domain.

Dixon and Kissinger (2013) demonstrated that span $\mathcal{L} \xleftarrow{i_1} \mathcal{I} \xrightarrow{i_2} \mathcal{R}$ adhering to Definition 5.3 uniquely defines the DPO transformation[6] of any linear string graph $\mathcal{G}$ to rewritten graph $\mathcal{G}'$ given matching $\mu : \mathcal{L} \to \mathcal{G}$, as depicted in (8).

$$\begin{array}{ccccc} \mathcal{L} & \xleftarrow{i_1} & \mathcal{I} & \xrightarrow{i_2} & \mathcal{R} \\ \mu \downarrow & & \mu' \downarrow & & \downarrow \\ \mathcal{G} & \longleftarrow & \mathcal{G} \setminus_{\mu} \mathcal{L} & \longrightarrow & \mathcal{G}' \end{array} \tag{8}$$

To embed pattern graphs in this rewrite rule, Definition 5.3 has to be extended. The extension utilises the notion of identity mapping $1_{\mathcal{L}}$ on object $\mathcal{L}$. An identity mapping (similar to the identify function) is the mapping of an object to itself, for example $1_{\mathcal{L}} : \mathcal{L} \to \mathcal{L}$, such that for any $f : \mathcal{L} \to \mathcal{G}$ it holds that $f \circ 1_{\mathcal{L}} = f = 1_{\mathcal{G}} \circ f$.

Definition 5.4* rewrite pattern: A span of pattern graphs $\mathcal{L} \xleftarrow{i_1} \mathcal{I} \xrightarrow{i_2} \mathcal{R}$ is a rewrite pattern, denoted, $\mathcal{L} \multimap \mathcal{R}$ if:

- $\mathcal{L} \setminus !_{\mathcal{L}} \xleftarrow{i_1} \mathcal{I} \setminus !_{\mathcal{I}} \xrightarrow{i_2} \mathcal{R} \setminus !_{\mathcal{R}}$ is a rewrite rule,
- for each $b \in !_{\mathcal{I}}$, $i_1(b)$ is the identity mapping of $i_2(b)$,
- the preimage[7] of $\mathcal{B}_{i_1(b)}$ under i_1 is $\mathcal{B}_b$, and
- the preimage of $\mathcal{B}_{i_2(b)}$ under i_2 is also $\mathcal{B}_b$.

The latter three conditions on Definition 5.4 ensure that each s!-box vertex present in $\mathcal{L}$ is present in $\mathcal{R}$ (although it might be dangling and implicitly removed). Also, its original representation before applying i_1, i_2 (the preimage) is one and the same s!-box. Furthermore, to support the s!-box operations within a rewrite pattern, each operation applied for the s!-box on the left hand side of a rewrite pattern, needs to be applied (in the same order) on the

[6] Dixon and Kissinger (2013) showed that for the category of string graphs (also known as open graphs) the pushout complement, which is the codomain of morphism $\mu' : \mathcal{I} \to \mathcal{G} \setminus \mathcal{L}_{\mu}$, exists and is unique, hence completes the pushout square.

[7] A preimage is the inverse of the image. It represents the original domain that resulted in the codomain after applying the morphism at hand.

right hand side. Let us now demonstrate that the sequential operations defined in Section 4.1 can be embedded into the presented rewrite pattern by combining each operation with the rewrite pattern span.

Definition 5.5: $\text{PsCOPY}_b(\mathcal{L} \multimap \mathcal{R})$ is the rewrite pattern equivalent of the original sequential copy operation. It is defined through the following span $\text{sCOPY}_{i_1'(b)}(\mathcal{L}) \xleftarrow{i_1'} \text{sCOPY}_b(\mathcal{J}) \xrightarrow{i_2'} \text{sCOPY}_{i_2'(b)}(\mathcal{R})$. The construction of morphism i_1' is a two-step approach. Firstly, (9) and (10) are original copy operations to construct $\text{sCOPY}_b(\mathcal{J})$ and $\text{sCOPY}_{i_1'(b)}(\mathcal{L})$:

$$\begin{array}{ccccc} \mathcal{B}_b & \longleftarrow & \text{Bound}(\mathcal{B}_b) & \longrightarrow & \mathcal{B}_{bb} \\ {\scriptstyle g}\downarrow & & \downarrow & & \downarrow \\ \mathcal{J} & \longleftarrow & \mathcal{J} \setminus_g \mathcal{B}_b & \longrightarrow & sCOPY_b(\mathcal{J}) \end{array} \tag{9}$$

$$\begin{array}{ccccc} \mathcal{B}_{i_1'(b)} & \longleftarrow & \text{Bound}(\mathcal{B}_{i_1'(b)}) & \longrightarrow & \mathcal{B}_{i_1'(b)i_1'(b)} \\ {\scriptstyle g}\downarrow & & \downarrow & & \downarrow \\ \mathcal{L} & \longleftarrow & \mathcal{L} \setminus_g \mathcal{B}_{i_1'(b)} & \longrightarrow & sCOPY_{i_1'(b)}(\mathcal{L}) \end{array} \tag{10}$$

Secondly, observe that $i_1 : \mathcal{J} \to \mathcal{L}$ in Definition 5.3 can be regarded as a matching. This in turn uniquely identifies the pushout complement $\mathcal{L} \setminus_{i_1} \mathcal{J}$ resulting in the DPO where i_1' completes the rewrite rule equivalent of the sequential copy operation like the following:

$$\begin{array}{ccccc} \mathcal{B}_b & \longleftarrow & \text{Bound}(\mathcal{B}_b) & \longrightarrow & \mathcal{B}_{bb} \\ {\scriptstyle g}\downarrow & & \downarrow & & \downarrow \\ \mathcal{J} & \longleftarrow & \mathcal{J} \setminus_g \mathcal{B}_b & \longrightarrow & sCOPY_b(\mathcal{J}) \\ {\scriptstyle i_1}\swarrow & & \swarrow & & \swarrow {\scriptstyle i_1'} \\ \mathcal{L} & \longleftarrow & \mathcal{L} \setminus_{i_1} \mathcal{J} & \longrightarrow & sCOPY_{i_1'(b)}(\mathcal{L}) \end{array} \tag{11}$$

One can similarly obtain morphism i_2' which uniquely defines $\text{sCOPY}_b(\mathcal{I}) \xrightarrow{i_2'} \text{sCOPY}_{i_2(b)}(\mathcal{R})$ and shows that the sequential copy operation is compatible with any rewrite pattern $\mathcal{L} \multimap \mathcal{R}$.

Definition 5.6: $\text{PsKILL}_b(\mathcal{L} \multimap \mathcal{R})$ is the rewrite pattern equivalent of the sequential kill operation defined by the span $\text{sKILL}_{i_1'(b)}(\mathcal{L}) \xleftarrow{i_1'} \text{sKILL}_b(\mathcal{I}) \xrightarrow{i_2'} \text{sKILL}_{i_2'(b)}(\mathcal{R})$. Morphisms i_1', i_2' are constructed analogous to Definition 5.5.

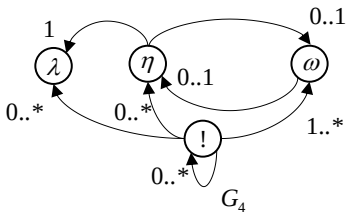
Definition 5.7*: $\text{PDROP}_b(\mathcal{L} \multimap \mathcal{R})$ is defined by the following span $\text{DROP}_{i_1'(b)}(\mathcal{L}) \xleftarrow{i_1'} \text{DROP}_b(\mathcal{I}) \xrightarrow{i_2'} \text{DROP}_{i_2'(b)}(\mathcal{R})$, such that i_1', i_2' are the restrictions of i_1, i_2 to $\text{DROP}_b(\mathcal{I})$.

This allows us to use rewrite rules where s!-boxes are part of the rule, or more formally; given some matching $\mu : \mathcal{L} \to \mathcal{G}$ with $\mathcal{G}$ being a linear string graph and $\mathcal{L}$ an instantiation, we can apply an identical instantiation sequence on rewrite pattern $\mathcal{L} \multimap \mathcal{R}$. This completes this algebra formalization enabling the reader to use this algebra to construct recursive rewrite patterns adopting the notation as shown initially in Figure 5(a), obviating the need for defining an infinite number of explicit rewrite rules.

5.2. Labelling regular vertices in (linear) string graphs

In the algebra presented so far, each regular vertex is treated equal. However, in applying the graph rewriting method in a practical context one typically has regular vertices (links) that have been assigned distinct labels. These labels can then be used to uniquely identify their behavior when applying a rewrite pattern. Labelling the regular vertices (links) in string graphs is isomorphic to introducing an additional vertex type λ connecting solely to regular vertices. Here, the epimorphism $lbl : \eta \to \lambda$ from a regular vertex to a label vertex

identifies the label on each regular vertex. The accompanying type graph G_4, including multiplicities, is given by:



(12)

For the purpose of the aggregation method in the next section we adopt (12) in a specific form where we have only two labels, the *critical-delay pattern graph*:

Definition 5.8 CDLSPATGraph: The category **CDLSPATGraph** of *critical-delay pattern graphs* is the full subcategory of the slice category Graph/G_4 adhering to the multiplicities indicated. In addition $|\lambda| = 2$, i.e. each regular vertex is mapped on one of two possible labels. One denoting a *critical-delay* vertex, the other a *non critical-delay* vertex.

6. Network aggregation methodology

As stated earlier, the reason of introducing rewrite patterns is to be able to (graphically) decompose the transport network into a free flowing, i.e. hypocritical, component and a queueing delayed, i.e. hypercritical, component, based on (1). Let us discuss how this can be achieved for each of the two travel time components.

6.1. Path travel time free flow decomposition

The hypocritical decomposition, is essentially an identity mapping on the full spatial representation of a path, i.e. one keeps the original path, but restricts itself to collecting only the hypocritical link travel times resulting in τ_p^I. Recall that this hypocritical component of

the path travel time is invariant to flow. Since a fixed path set P is used, one can simply compute τ_p^I once for each path, store it, and reuse it for each demand scenario s. We note that this approach can only be lossless under the assumption that $\tau_p^{II(s)}\left(f_p^{(s)}\right)=0$, because if it isn't the hypocritical delay is no longer invariant to flow and it can no longer be pre-computed for all demand scenarios.

6.2. Path travel time hypercritical delay decomposition

Given (1) it is clear that links which, under any of the considered demand scenarios $s \in S$, do not contribute to constructing the hypercritical path delay $\tau_p^{III(s)}\left(f_p^{(s)}\right)$, can be removed from the paths traversing it. Doing so incurs no information loss on the part of the path travel times and therefore classifies as lossless. Conversely, links that do exhibit delay for any of the considered demand scenarios should be kept. We point out that in some cases links are to be kept even if they exhibit no direct delay. For example, in traffic assignment models including a node model, delay is typically imposed through the node model which in turn is dependent on desired flow rates from its incoming links and available supply on its outgoing links. Therefore, in case any of the incoming links experience delay imposed by the node model, the entire spatial topology of the node needs to be retained, including the non-delayed incoming and outgoing links, to be able to obtain identical delays in the delay decomposed network.

Based on this observation, without making any further assumptions on the underlying traffic assignment procedure, a single rewrite pattern is presented that is capable of converting any original path into its delay decomposed counterpart. This however, does require partitioning each link into one of two types: links critical to reconstructing the delay, i.e. *critical-delay links*, and links that are not critical, i.e. *non critical-delay links*.

6.3. Network aggregation rewrite pattern

In the remainder of this work, λ vertices (Definition 5.8) are implicitly included in the graph representation by colouring regular vertices, see Figure 8(a).

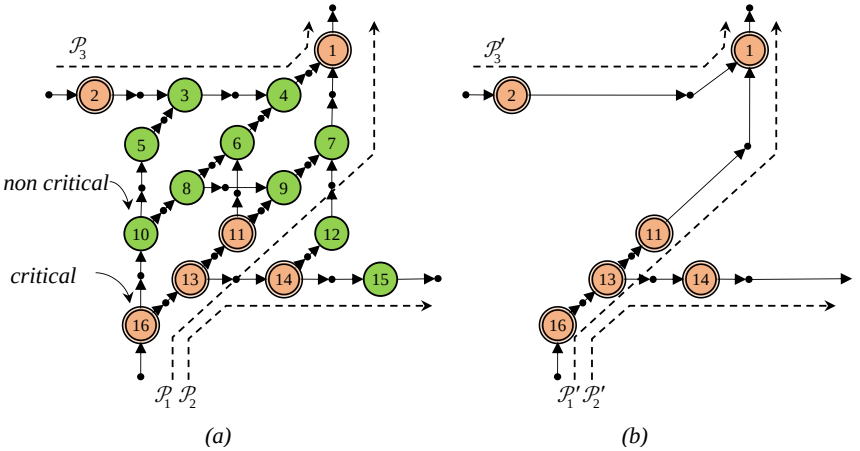
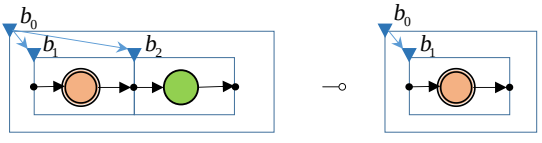


Fig. 8. (a) Critical-delay labelled example network with three example paths. (b) Delay decomposed network induced after applying path based rewrite rule.

The goal is now to obtain the delay decomposed network depicted in Figure 8(b) which only contains the delay critical links and the abstracted connections between them in case links in between have been removed. A two-step approach is adopted, firstly all original paths $\mathcal{P}_p$ are rewritten to their delay decomposed counterparts $\mathcal{P}_p{}'$ through the single rewrite pattern given in (13):



(13)

This rewrite pattern tries to match zero or more (sequential) *critical-delay links* followed by zero or more (sequential) *non critical-delay links* and this itself can be repeated zero or more times. The match found it is rewritten to the same sequence except that all *non critical-*

delay links have been removed. Figure 9 gives an example of how this works practice (in single pushout notation). Notice how both $\mathcal{P}_p$ and $\mathcal{P}_p'$ are instantiations of the left and right hand side of the rewrite pattern, respectively.

The second step entails constructing the entire delay decomposed transport network as a simplified string graph. Since each linear string graph is a special case of a simplified string graph, one can simply take the union of all delay decomposed paths $\mathcal{P}_p'$ to obtain delay decomposed transport network $\mathcal{G}'$ as per (14).

$$\mathcal{G}' = \bigcup_{p \in P} \mathcal{P}_p' \tag{14}$$

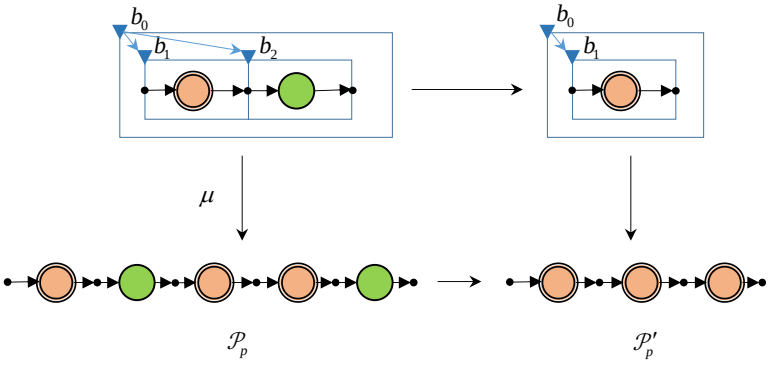


Fig. 9. Example matching and rewrite of a path into its delay decomposed counterpart.

6.4. Aggregation procedure properties

The proposed method remains lossless as long as all vertices identified as delay-critical in any of the considered demand scenarios are included in the delay decomposed network. This suggests requiring knowledge on the equilibrium solution for the labelling. However, the purpose of this method is to speed up the traffic assignment procedure. If for every scenario the equilibrium solution is needed before one can aggregate no computational gain will be achieved. To circumvent this issue, the aim is to construct a *single* delay decomposed

network from equilibrating a single *'super scenario'* s^* that contains all (potentially) critical-delay vertices, i.e. s^* is lossless for all considered scenarios $s \in S$.

First, observe that a naïve solution for constructing the supply network for s is found by simply retaining all links by marking them as delay-critical. Clearly, this would not result in any computational gain. Let us now make the additional assumption that the hypercritical path delay $\tau_p^{III(s)}(f_p^{(s)})$ is a link additive, strictly increasing function. Then one can construct s^* given $f_p^{(s^*)} \geq f_p^{(s)}, \forall s \in S, \forall p \in P$. Since $\tau_p^{III(s)}(f_p^{(s)})$ is strictly increasing, each vertex that would be identified as delay-critical in any s, must by definition also be in s^*, because if not, it would contradict with the strict monotonicity of $\tau_p^{III(s)}(f_p^{(s)})$. Whenever any of these assumptions are violated the proposed method might no longer be lossless. The authors point out that in a practical context, even when there might be information loss, such a case one can easily be verified by comparing the aggregated result to the original network scenario combination. One can even incrementally build s^*, by subsequently including missing delay-critical vertices while computing the various demand scenarios.

7. Delay decomposed path reduction

The proposed network aggregation method in Section 6 can be extended to increase its efficiency. This extension identifies the minimal set of delay composed paths needed in traffic assignment, while maintaining the original path delays. The effectiveness of this method relies on the significantly increased likelihood of path overlap in a delay decomposed network.

Let us illustrate this with the example presented in Figure 10 (a-c). None of the original paths in Figure 10(a) overlap completely. Assuming Figure 10(b) represents the found '*super-scenario*' we obtain the delay decomposed network as presented in Figure 10(c) based on (13) and (14). At this stage the first two paths and last two paths now overlap completely and can be merged to reduce the path set needed in traffic assignment. One only

needs to make sure that the demands match and a mapping is maintained in order to reconstruct identical path delays.

Formally, we need to somehow be able to state that $\mathcal{P}_3'$ and $\mathcal{P}_4'$ are considered equivalent as well as $\mathcal{P}_1'$ and $\mathcal{P}_2'$. This can be achieved through the definition of an equivalence relation and accompanying quotient mapping which is a standard approach of modelling such relations.

Definition 7.1 delay decomposed path equivalence: Let us define equivalence relation $\mathcal{R}^{(s)} \subseteq P \times P$ for a demand scenario s. $\mathcal{R}^{(s)}$ is defined such that any two paths $p, p' \in P$ are considered equivalent given that morphism $g : \mathcal{P}_p'^{(s)} \rightarrow \mathcal{P}_{p'}'^{(s)}$ is the identity mapping, i.e. whenever two paths have the exact same representation in their delay decomposed form they are considered equivalent under $\mathcal{R}^{(s)}$.

Using equivalence relation $\mathcal{R}^{(s)}$ of Definition 7.1 one simply partitions all paths into their respective equivalence classes. This is known as the quotient mapping induced by equivalence relation $\mathcal{R}^{(s)}$, denoted $P / \mathcal{R}^{(s)}$. One of the benefits of this approach is that it yields projection $\pi^{(s)} : P \rightarrow P / \mathcal{R}^{(s)}$ which maps each path $p \in P$ to its respective equivalence class based on $\mathcal{R}^{(s)}$.

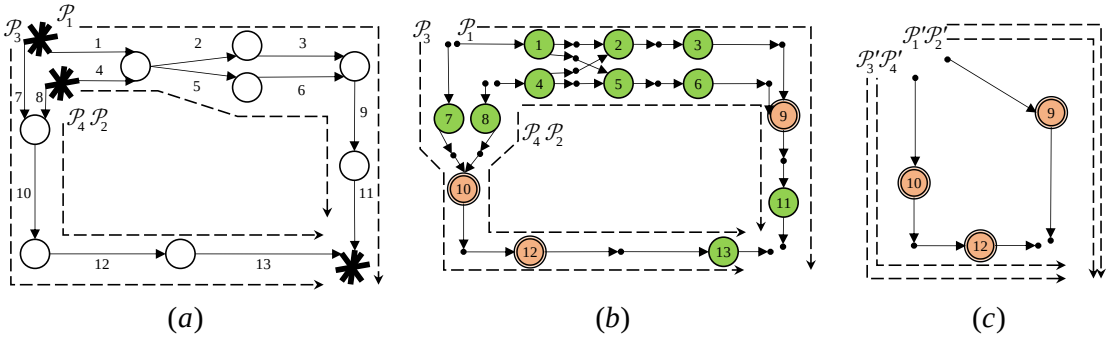


Fig. 10. (a) Original network, with four paths, (b) critical-delay labelled string graph representation of the same network, (c) delay-decomposed network induced from delay decomposed paths.

Consider the example in Figure 10(a) with $P = \{p_1, p_2, p_3, p_4\}$. Given labelling under scenario s^* (Figure 11(b)) the delay decomposed representations $\mathcal{P}'_{1-4}$ of these paths are depicted in Figure 10(c). The equivalence relation $\mathcal{R}^{(s)}$ under this specific example then yields $\mathcal{R}^{(s)} = \{(p_1, p_2), (p_2, p_1), (p_3, p_4), (p_4, p_3)\}$. Clearly, this results in two equivalence classes. We adopt standard notation for equivalence classes which results in the following, the projections from original path to equivalence class, i.e. $\pi^{(s)}(p_1) = [p_1] = \pi^{(s)}(p_2) = [p_2]$ and $\pi^{(s)}(p_3) = [p_3] = \pi^{(s)}(p_4) = [p_4]$, where $[p_1]$ and $[p_2]$ represent the same equivalence class, as do $[p_3]$ and $[p_4]$. The number of unique paths on the delay decomposed network automatically follows from the cardinality of available equivalence classes, i.e. $|P / \mathcal{R}_s|$. This is also the minimal number of paths one needs in the assignment to obtain the original path delays. The path flow for each distinct path represented by an equivalence class has become a trivial task and is simply given by:

$$f_{[p]}^{(s)} = \sum_{p' \in [p]} f_{p'}^{(s)}, \quad [p] \in P / \mathcal{R}^{(s)}, \tag{15}$$

where $[p]$ is the equivalence class at hand and $p' \in [p]$ are all paths that project onto the same delay decomposed path. When one applies this path aggregation procedure on top of the delay decomposition method computation time can be reduced significantly, since the computational burden is fully path driven. Note that the original travel time delay function needs to be altered slightly since the delay component is obtained by mapping it to its equivalence class:

$$c_p^{(s)} = \tau_p^I + \tau_{[p]}^{III(s)}\left(f_{[p]}^{(s)}\right), \quad [p] = \pi^{(s)}(p), p \in P, s \in S. \tag{16}$$

On a final note, we like to point out that a similar mapping is needed for the underlying route choice procedure of the adopted traffic assignment methodology, because each unique equivalence path can represent paths with multiple origins and/or destinations.

8. Case studies

To demonstrate the applicability and effectiveness of the proposed aggregation methods, a hypothetical and large scale case study is conducted.

8.1. Traffic assignment procedure

In this case study the path-based quasi-dynamic traffic assignment model proposed by Bliemer et al. (2014a) is adopted. This model is compliant with (1)[8]. Furthermore, it has been shown (Bliemer et al. (2014b)) that it can be directly derived from the (dynamic) Generalised Link Transmission Model (Gentile, 2011), which should make it arguably more realistic than its traditional static peers. This derivation is possible under the following assumptions: (i) a concave hypocritical branch of the fundamental diagram and a horizontal hypercritical branch, (ii) an infinite link storage capacity, i.e. no spillback leading to vertical queues and (iii) a proper first order node model. To satisfy (iii) the node model by Tampere et al (2011) is adopted dictating the distribution of accepted flows for merges, diverges, and more general cross-nodes. The undelayed path travel time in this model is given by:

$$\tau_p^I = \sum_{v \in \eta_{\mathcal{P}_p}} \frac{L_v}{\vartheta_v}, \quad p \in P, \tag{17}$$

[8] The underlying travel time function in Bliemer et al. (2014) while compliant with (1) is neither link additive nor strictly increasing. As a result it is possible that super scenario S^* does not result in a lossless aggregation result under varying demand. However, this case study only considers a single demand scenario to illustrate the potential computational gain. In such a testbed the procedure is lossless, since demand does not vary.

with L_η and ϑ_η representing the length and free speed of the road segment represented by vertex $\eta \in \eta_{\mathcal{P}_p}$, respectively. Whenever some of the flow is held back by the node model a point queue forms. Link queue magnitudes are node model controlled by imposing a reduction factor $\alpha < 1$. Conversely, when a link has no queue $\alpha = 1$. This reduction factor α determines the delay. It can be shown that the total path delay is given by:

$$\tau_p^{III(s)}\left(f_p^{(s)}\right) = \frac{T}{2}\left(\frac{1}{\prod_{v\in\eta_{\mathcal{P}_p}} \alpha_v^{(s)}} - 1\right), \quad p \in P, s \in S, \tag{18}$$

where T is the total simulation time. Since queues grow linearly over time, dividing by two results in the average delay. For exact details on the network loading we refer to Bliemer et al. (2014).

8.2. Hypothetical corridor example

To illustrate the combined result of applying the presented aggregation method in coherence with the traffic assignment model of Bliemer et al. (2014) a corridor network is presented (Figure 11(a)). It contains a single path p, with $f_p = 3000$ (veh/h) and $T = 1h$ A corridor simplifies the node model, i.e. the (turn) reduction factor computation to $\alpha_{vv'} = \min\{1, Q_{v'}/q_{vv'}\}$, with outgoing link capacity $Q_{v'}$ and desired turn flow rate $q_{vv'}$ (from incoming link v to outgoing link $v, \{v, v'\} \in \eta_{\mathcal{P}_p}$). Clearly, the first four links can accommodate the desired flow, resulting in $\alpha_{1-4} = 1$, i.e. no reduction. Given our node model $\alpha_{5,6} = \min\{1, 2000/3000\} = 2/3$. The flow offered at the end of link v_6 is obtained by multiplying the desired flow rate with the link reduction factor, i.e. $q_{6,7} = 3000 \cdot (2/3) = 2000$. Applying the node model again yields $\alpha_{6,7} = \min\{1, 1000/2000\} = 1/2$, which provides the flows on links $v_{7-10} = 1000$ veh/h reaching the destination. The 2000 vehicles that departed but are stuck in the network emerge as point queues. Recall that queues are assumed to grow

linearly in time. Thus, the first vehicle on a link does not encounter a queue, while the last vehicle experiences a queue length of $q_{vv'}(1-\alpha_{vv'})$.

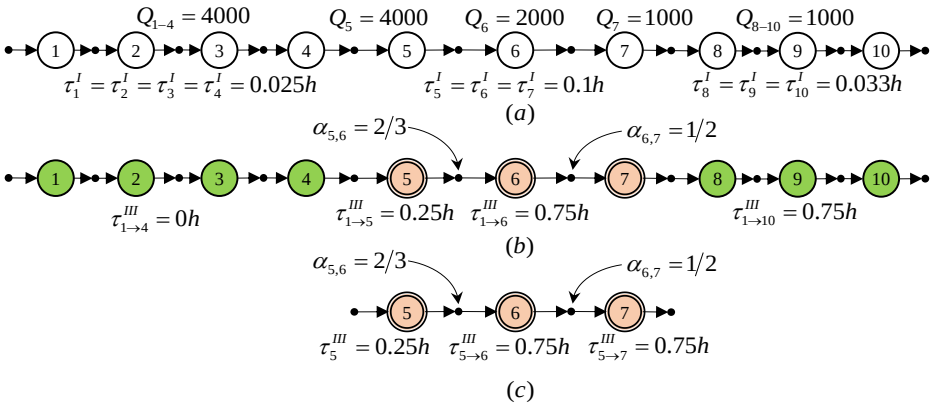
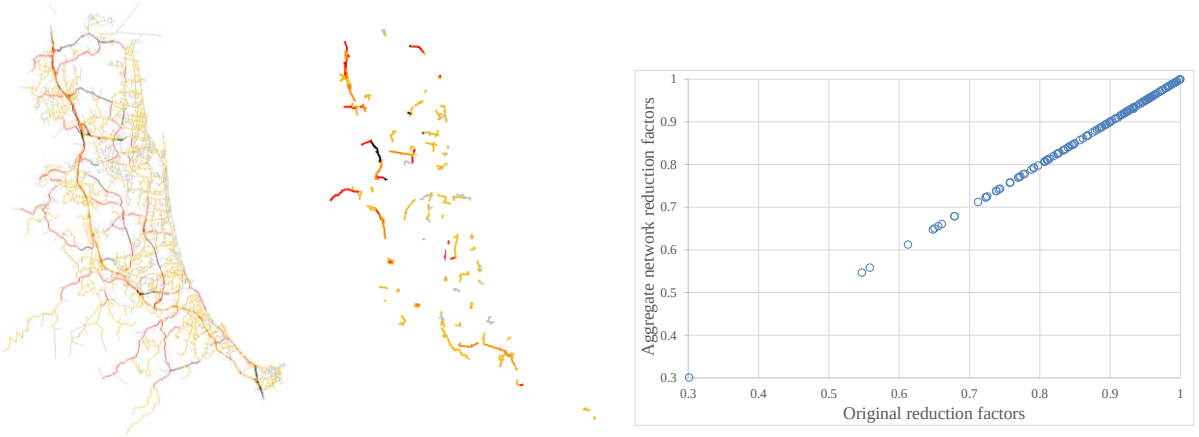


Fig. 11. (a) Corridor network with link capacities and (additive) non-delay link travel times. (b) Result under $f_p = 3000$ ***veh/hr, reduction factors and cumulative delays from the origin up to the current link. (c) Delay decomposed network after identifying critical delay links yielding identical delay travel time.***

The link delay shown in Figure 11(b) is computed via (18), note that the cumulative delay is shown and not the individual link delay which amounts to an *average* delay of fourty-five minutes, i.e. $3/4h = 1/2\left(1/\left((2/3)\cdot(1/2)\right)-1\right)$. The observant reader notices that this delay is different (and arguably more realistic) compared to the typical additive delay functions that follow the work of Payne and Thompson (1975). Finally, observe that link 7 is also labelled as a critical-delay link. This is because it is required to be able to correctly determine the reduction factor on the previous link.

8.3. Large scale case study: Gold Coast

In this section, the two aggregation procedures are sequentially applied to the large-scale network of Gold Coast, Queensland Australia as depicted in Figure 12(a).



(a) *(b)* *(c)*

Fig. 12. (a) Original Gold Coast network, (b) Gold Coast network after delay-decomposition and zonal redistribution, (c) comparing reduction factors between original and aggregate equilibrium result of 80 iterations.

This network is kindly provided by Veitch-Lister Consultancy and subsequently made publicly available via Bar-Gera (2016). This network contains: 2987 nodes, 5076 (bi-directional) links, 1067 origins and 1067 destinations. A hypothetical (one hour) morning peak origin-destination matrix is utilised, containing roughly 120,000 veh/h. This demand is used to represent *'super-scenario' s^**.

In this case study a Stochastic User equilibrium (SUE) approach is adopted (Daganzo and Sheffi, 1977), i.e. perceived cost is used instead of experienced cost. Regarding route choice a conditional logit model (McFadden, 1973) is used. The route choice scaling parameter, defining the level of uncertainty in the available information, is set to 5, see also Chen (1999). Path set *P* is generated a-priori, based on Fiorenzo-Catalano et al. (2004): Resulting in a conditional SUE. The Method of Successive Averages (MSA) is used to smooth iteration results, using a Polyak (1990) inspired smoothing factor of 0.7. The proposed aggregation procedures have been implemented as prototypes within the StreamLine framework[9].

[9] StreamLine is a Dynamic Traffic Assignment framework underpinning all non-static assignment procedures in the OmniTRANS transport planning software suite, kindly provided by DAT.mobility.

After equilibrating the original network the network delay decomposition method is applied and subsequently the minimal delay decomposed path set is constructed. Figure 12(b) shows the delay decomposed network induced from the delay decomposed paths.To illustrate that the equilibrated aggregate network yields the same result as the original network, the reduction factors of two assignments, one on the original network and the other on the delay decomposed delay network are compared: Firstly by creating a 45-degree plot, see Figure 12(c). Secondly, we examine the norm $\|\tilde{\alpha}_v\| = 0.00366,$, which is defined as:

$$\|\tilde{\alpha}_v\| = \left(\sum\nolimits_{v \in \eta_{\mathcal{G}}} \left(\alpha_v^{\Delta} \right)^2 \right), \tag{19}$$

with α_v^{Δ} denoting the difference in reduction factor value between the original network $\mathcal{G}$ and the delay decomposed network for link v. The effect on computation times when applying the proposed aggregation procedure(s) turns out to be significant as outlined in Table 2.

Table 2. Effects of aggregation on computation times and topology (80 iterations).

Network	Links	Paths	Total path links	Total network loading time (s)
Original	5076	1,221,446	55,852,786	2178.48
Delay-decomposition	836	1,221,446	12,919,862	902.57 *(-59%)*
Delay-decomposed path set aggregation	836	99,528	1,142,338	88.62 *(-96%)*

The computational burden is reduced by an order of magnitude, due to the decrease in both size and number of paths that need to be loaded in each iteration. Applying only delay decomposition halves network loading times on this network. This can be attributed to the reduced number of links per path (10.6 vs 45.7 links on average). However, it is not until the combined procedure is employed that the method's full potential is revealed. The minimal path set contains less than 10% of the original (<100,000 vs 1,221,446) paths due to the increased amount of overlap. As a result, the time required to complete network loading drops to 4.1% of the original computation time, meaning one can roughly compute

up to twenty-five demand scenarios in a similar amount of time it takes to compute a single scenario on the original network.

9. Conclusions and further research

In this work a category theory based formalisation method is introduced to the field of transport. This method is then employed to present two novel spatial aggregation procedures compatible with a class of traffic assignment models complying with equation (1). The introduced sequential pattern graph rewriting technique is deliberately kept fairly generic, so it can be reused to describe other aggregation methods than the one presented here. Due to its close resemblance to functional programming each rewrite rule can almost directly be converted into an equivalent recursive algorithm which first matches against a graph and subsequently rewrites it, based on the sequence of operations identified.

The two aggregation methods specifically target a range of applications where the infrastructure, i.e. supply side is kept fixed while demand varies. Under the assumption that travel time is link additive and strictly increasing the proposed procedure is lossless.Practical implications of applying the presented aggregation procedure are threefold: (i) Practitioners can either consider more scenarios in the same amount of time, which improves the reliability and robustness of recommendations made or, (ii) practitioners have more time investigating the subset of scenarios that require further investigation, improving the accuracy of the studies. (iii) One can also opt to adopt a more sophisticated assignment method that yields more realistic results. The additional time it used to take to adopt such a model, like for example the Bliemer et al. (2014) model used in the Gold Coast case study, is mitigated by this aggregation procedure.

Results on the Gold Coast case study show that network loading can be reduced significantly (4.1% of the original network loading time) depending on the employed network topology and demand scenarios. This is achieved by a network delay decomposition method that reduces path and network topology and a subsequent path reduction method that identifies and removes overlapping delay-decomposed paths such that only a minimal set of paths is needed in assignment.

9.1. Further research

The proposed procedure is guaranteed to be lossless when a *'super-scenario' s** can be identified. While one might expect this is the scenario with the highest demand, in practice this might not always the case. Models, like the Bliemer at al. (2014) model, that do not have monotonic increasing cost functions do not necessarily generate more critical-delay links with higher demand. Identifying the *'super-scenario'* is therefore not trivial. Further research is needed to: (i) See if a procedure exists that quickly identifies the *'super-scenario'* for models with non-monotonic cost functions, (ii) in case no *'super-scenario'* is identified, estimate the amount of information loss suffered when still applying the proposed aggregation procedure.

It would also be of interest to investigate the effects of this aggregation procedure on other compatible traffic assignment models and compare then. This would provide further insight in the robustness and generalisability of the methodology.

References

Awodey, S., (2010). Category Theory, Second edi. Category theory. Oxford University Press.

Bar-Gera, H., (2016). Transportation test problems. https://github.com/bstabler/TransportationNetworks.

Barton, R.R. and Hearn, D., 1979. Network aggregation in transportation planning models - final report.

Bliemer, M.C.J. Raadsen, M.P.H., Smits, E-S. Zhou, B. Bell, M.G.H. (2014a). Quasi-dynamic traffic assignment with residual point queues incorporating a first order node model. Transportation Research Part B: Methodological, 68, pp.363–384.

Bliemer, M.C.J., Raadsen, M.P.H., Brederode, L.J.N., Bell, M.G.H., Wismans. (2014b). A unified framework for traffic assignment: deriving static and quasi-dynamic models consistent with general first order dynamic traffic assignment models. 5th Symposium on dynamic traffic assignment, Salerno, Itlay.

Bovy, P. and Jansen, G. (1983). Network aggregation effects upon equilibrium assignment outcomes: An empirical investigation. Transportation Science, 17(3), pp.240–261.

Bureau of Public Roads, United States (1964). Traffic assignment manual, Washington.

Cascetta, E. (2009). Transportation systems analysis: Models and applications (2nd ed.). New York, NY: Springer.

Chan, Y., Follensbee, K.G., Manheim, M.L., Mumford, J.R. (1968). Aggregation in transport networks: An Application of Hierarchical Structure., M.I.T. dept. Civil Eng. Res. Report No R68-47.

Chan, Y., 1976. A method to simplify network representation in transportation planning. Transportation Research, 10(3), pp.179–191.

Chang, K.T., Khatib, Z., Ou, Y. (2002). Effects of zoning structure and network detail on traffic demand modeling. Environment and Planning B: Planning and Design, 29, pp.37–52.

Chen, H-K. (1999). Dynamic travel choice models, a variational inequality approach, Berlin: Springer-Verlag.

Coecke, B. and Duncan, R. (2011). Interacting quantum observables: categorical algebra and diagrammatics, New J. Phys. 13 (2011).

Connors, R.D. and Watling, D.P. (2008). Aggregation of Transport Networks Using Sensitivity Analysis. In Proceedings of the European Transport Conference. Association for European Transport.

Daganzo, C.F. and Sheffi, Y. (1977). On Stochastic Models of Traffic Assignment. Transportation Science, 11(3), pp.253–274.

Daganzo, C. F. (1980a). An equilibrium algorithm for the spatial aggregation problem of traffic assignment. Transportation Research Part B: Methodological, 14(3), pp.221–228.

Daganzo C.F. (1980b). Network representation, continuum approximations and a solution to the spatial aggregation problem of traffic assignment. Transportation Research Part B: Methodological, 14(3), pp.229–239.

Dixon, L., Duncan, R., (2009). Graphical reasoning in compact closed categories for quantum computation. Ann. Math. Artif. Intell. 56, 23–42.

Dixon, L., Duncan, R., Kissinger, A., (2010). Open Graphs and Computational Reasoning. Dev. Comput. Model. Causality, Comput. Phys. 26, 169–180.

Dixon, L., Kissinger, A., (2013). Open-graphs and monoidal theories. Math. Struct. Comput. Sci. 23, 308–359. doi:10.1017/S0960129512000138.

Ehrig, H., Pfender, M., Schneider, H.J., (1973). Graph-grammars: An algebraic approach. 14th Annu. Symp. Switch. Autom. Theory (swat 1973).

Fiorenzo-Catalano, S., van Nes, R., Bovy, P.H.L. (2004). Choice Set Generation for Multi-modal Travel Analysis. European Journal of Transport and Infrastructure Research, 4(2), pp.195–209.

Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. Naval Research Logistics quarterly, 3, pp.95–109.

Gentile, G. (2011). The General Link Transmission Model for Dynamic Network Loading and a comparison with the DUE algorithm. In New Developments in Transport Planning: Advances in Dynamic Traffic Assignment (Chapter 8).

Hearn, D. W. (1984) Practical and theoretical aspects of aggregation problems in transportation planning models. Transportation Planning Models. New York, Elsevier.

Joyal, A., Street, R., (1991). The geometry of tensor calculus, I. Adv. Math. (N. Y). 88, 55–112

Jeon, J-H., S-Y. Kho, J.J. Park, D-K. Kim (2012). Effects of spatial aggregation level on an urban transportation planning model. KSCE Journal of Civil Engineering, 16, pp.835–844.

Kissinger, A., Merry, A., Soloviev, M., (2014). Pattern graph rewrite systems. Electron. Proc. Theor. Comput. Sci. 143, 54–66.

Long, G.D. and Stover, V.G. (1967). The effect of network detail on traffic assignment results. Research Report Number 60-11, Texas Transportation Institute, College Station, Texas.

Löwe, M., (1993). Algebraic approach to single-pushout graph transformation. Theor. Comput. Sci. 109, 181–224.

McFadden, D. (1973). Conditional logit analysis of qualitative choice behavior. In Frontiers in Econometrics. pp. 105–142.

Newell G.F. (1979). Traffic flow on transportation networks, Cambridge Massachusetts: MIT Press.

Newell G.F. (1993). A Simplified Theory of Kinematic Waves In highway Traffic, Part II: Queuieng At Freeway Bottlenecks. Transportation Research Part B: methodological. Volume 27B issue 4, 1993, pp.289-303.

Openshaw, S. and Taylor, P.J., (1979). A million or so correlation coefficients: three experiments on the modifiable areal unit problem. statistical applications in spatial sciences, pp.44–127.

Osorio, C. and Bierlaire, M., (2009). An analytic finite capacity queueing network model capturing the propagation of congestion and blocking. European Journal of Operational Research, 196(3), pp.996–1007.

Páez, A. and Scott, M., (2004). Spatial statistics for urban analysis: A review of techniques with examples. GeoJournal, 61, pp.53–67.

Payne, H.J., and Thompson, W.A. (1975). Traffic assignment on transportation network with capacity constraints and queuing. 47th National ORSA/TIMS North Am. Meeting.

Polyak, B.K. (1990). New method of stochastic approximation type. Automated Remote Control, 51(7), pp.937–946.

Simmons, H., (2004). An Introduction to Category Theory. Univ. Connect.

Tampère, C.M.J. Corthout, R.,Cattrysse, D., Immers, L.H. (2011). A generic class of first order node models for dynamic macroscopic simulation of traffic flows. Transportation Research Part B: Methodological, 45(1), pp.289–309.

Vickrey, W.S., (1969). Congestion theory and transport investment. American Economic Review, Papers and Proceedings 59, 251-260