Route Choice in Stochastic Time-Dependent Networks

Theory, Algorithms and Computational Experience

Lars Relund Nielsen

PhD thesis, December 2003

Route Choice in Stochastic Time-Dependent Networks

Theory, Algorithms and Computational Experience

Lars Relund Nielsen, Department of Operations Research,
University of Aarhus, Denmark

PhD thesis, December 2003

Thesis committee:

- Jørgen Aase Nielsen, University of Aarhus.
- Jørgen Tind, University of Copenhagen.
- Kurt O. Jörnsten, Norwegian School of Economics and Business Administration.

Thesis advisor:

- Kim Allan Andersen, Aarhus School of Business.

Lars Relund Nielsen
Department of Operations Research
University of Aarhus
Ny Munkegade, building 530
DK-8000 Aarhus C
Denmark
relund@imf.au.dk
http://home.imf.au.dk/relund/

Subject classification

MSC2000 – Primary: 90B15; secondary: 90B50, 90B06, 90C47, 90C35, 90C15, 90C59.
OR/MS – Networks/graphs: generalized networks, stochastic; transportation: network models, route choice.

Dedicated to Dorthe and Mads

For angling may be said to be so like the Mathematics, that it can never be fully learnt; at least not so fully, but that there will still be more new experiments left for the trial of other men that succeeds us.

Izaak Walton,
The Compleat Angler, 1653

Preface

This thesis is the outcome of my research during my four years as a PhD student at the Department of Operations Research at the University of Aarhus. The main field of my research has been directed hypergraphs and how they can be used to model problems in stochastic time-dependent networks (*STD networks*). Other areas such as logical inference and logic-based methods for optimization have also gained some interest, but will not be covered by this thesis.

As the title suggests, the thesis focuses on STD networks which are an extension of more "traditional" networks where the travel time or cost between two nodes (towns, telephone switches etc.) are deterministic and time-independent. In STD networks the travel time between two nodes are time-dependent, i.e. the travel time depends on the leaving time from a node. Furthermore, it is assumed that, for each leaving time, the travel time may not be fully known and hence a probability function is used to express possible travel times. This gives in many cases a better framework for modelling real world problems.

We consider route choice problems in STD networks which may be regarded as extensions of traditional shortest path problems in directed graphs. The problem of finding a shortest path in a directed graph may be considered as two problems in an STD network, depending on whether the entire route, denoted a *strategy*, must be specified a priori, i.e. before travel begins (*a priori route choice*) or whether the driver is allowed to react while travelling on the revealed/actual arrival times at intermediate nodes (*time-adaptive route choice*). The problem of finding the best route/strategy under a priori or time-adaptive route choice consists in finding a strategy which is minimal with respect to a specific objective, e.g. expected travel time.

The thesis focuses on two route choice problems in STD networks. In Chapter 4 we consider the problem of finding the K best strategies under a priori and time-adaptive route choice while Chapter 5 considers bicriterion route choice under a priori and time-adaptive route choice. Here we assume that two criteria are given, e.g. minimizing expected travel time and cost. The goal is now to find efficient strategies, i.e. strategies for which it is not possible to find a different strategy such that expected travel time or cost is improved without getting a worse expected cost or travel time, respectively. Finally, Chapter 6 presents two problems which, due to time issues, have not been studied as deeply as the problems in Chapter 4 and 5. That is, the results given may not be considered as complete but may be regarded as directions for further research.

All the problems in this thesis have either not been considered before in the literature or considered in a very limited number of papers.

How to read this thesis

First of all the reader is assumed to be familiar with graph theory and a few basic concepts from probability theory. The thesis is best read in chronological order. Theory and notation given in chapters preceding the chapter under consideration will often be used. However, there are exceptions. Chapter 1 presents a very short introduction to the problems we consider and a summary of the thesis which may be skipped. The appendices contain information not directly related to STD networks. Appendix A considers problems which have emerged as a result of studying the problem of finding the K best strategies in an STD network, namely the subhypertree constrained hyperpath problem and the problem of finding the K minimum weight hyperpaths in a non-acyclic hypergraph. Appendix B describes basic data structures used in the algorithms described in the thesis. Finally, during fall 2000 I found a small error in the definition of a hyperpath which has been used in many papers. I therefore wrote a short note with a counter example which is presented in Appendix C.

Since most of the problems considered in this thesis have either not been considered before we in most cases, had to start from scratch defining new notation, problem formulation etc. As a result the reader may find the notation heavy and comprehensive. Therefore the index "List of notation" has been included at the end of this thesis. It contains all the notation used in the thesis, a short explanation and a reference to the page where the notation is first introduced.

Acknowledgments

I owe a debt of gratitude to many people who have been crucial to my success in completing the four years of my PhD.

First of all, I am thankful to my supervisor Kim Allan Andersen for his guidance, encouragement, contributions and suggestions during the last four years.

During the late fall 2000 I visited Professor Daniele Pretolani at the University of Camerino, Italy. During three valuable months, I had the privilege to draw on his comprehensive knowledge on the field of directed hypergraphs. Furthermore, I would like to express my gratitude to him for our many discussions about solution methods and implementation details. My research has greatly benefited from this and I am deeply indebted.

I would like to thank my colleagues at the Department of Operations Research for always providing a pleasant working atmosphere. In particular I am deeply indebted to Randi Mosegaard for all the jokes (via e-mail) and for carefully proof reading the thesis.

Special gratitude is also extended to Holger and Søren for countless non-math hunting and fishing trips during the last four years helping clearing my mind.

Finally, I owe much to my parents for always believing in me and encouraging me to achieve my goals. My heartfelt appreciation also goes out to my wife Dorthe for her understanding and her steadfast emotional support which have been invaluable in helping me during this writing process.

Aarhus, December, 2003

Lars Relund Nielsen

Contents

Chapter 1 Introduction and summary

This thesis focuses on problems concerning route choice in stochastic time-dependent networks (*STD networks*) and is the result of merging quite different research fields. In this chapter we present a short introduction to the problems and the theory considered in this thesis. The chapter is concluded with a summary of the remainder of the thesis.

STD networks are an extension of more "traditional" networks where the travel time or cost between two nodes (towns, telephone switches etc.) is deterministic and time-independent. We say that a network is *time-dependent* if the travel times on the arcs are functions of time, and *stochastic* if the travel time is represented by probability distributions rather than simple scalars.

STD networks were first addressed by Hall [37] who considered the problem of finding a path between two nodes minimizing the expected travel time when leaving the origin at a specific time. He pointed out several ways to formulate the path selection problem in STD networks as well as complications arising as a consequence of modelling both the stochastic and time-dependent properties in the network.

If the entire route, denoted a strategy, must be specified before travel begins, the driver must follow a loopless path in the network and no deviations from the path are permitted. In this case the route is selected a priori on the basis of only the probability distributions of the arc travel-times. This is referred to as *a priori route choice* and may be useful for routing highly sensitive substances for which the path travelled must be preapproved or where the driver does not have access to (or time to access) information while travelling.

Rather than determining a single path only based on information known before travel begins, routes/strategies with lower expected travel time may be obtained by allowing the driver to react to revealed (actual) arrival times at intermediate nodes. This is identical to a multistage recourse problem where decisions are taken according to realizations of arc travel time outcomes while travelling (see Birge and Louveaux [9]). Hall realized that the best route minimizing expected travel time does not necessarily define a path in the network but rather a time-adaptive strategy that assigns optimal successors to a node as a function of time. This is referred to as *time-adaptive route choice*.

The problem of finding the best route/strategy under a priori and time-adaptive route choice consists in finding a strategy which is minimal with respect to a specific objective,

e.g. expected travel time. The problem under a priori route choice is $\mathcal{NP}$-hard while the problem under time-adaptive route choice can be solved in polynomial time (see Pretolani [77]). Route choice problems under both a priori and time-adaptive route choice will be considered in the thesis.

STD networks can be modelled using directed hypergraphs mainly used in the field of computer science. Directed hypergraphs are an extension of directed graphs and undirected hypergraphs and represent a general modelling and algorithmic tool, which have been successfully used in many different research areas such as artificial intelligence, database systems, fuzzy systems, propositional logic and transportation networks.

Pretolani [77] presented a hypergraph model for an STD network and showed that finding the best strategy under time-adaptive route choice corresponds to finding a minimum weight hyperpath in a time-expanded hypergraph. The definition of a hyperpath in a directed hypergraph and a general formulation of the minimum weight hyperpath problem were given by Gallo, Longo, Pallottino, and Nguyen [29].

This thesis focuses on two route choice problems in STD networks which may be regarded as extensions of traditional shortest path problems in directed graphs. In Chapter 4 we consider the problem of finding the K best strategies under a priori and time-adaptive route choice. That is, ranking the first K best strategies defining the route followed between an origin and a destination node when leaving the origin at a specific time, in non-decreasing order with respect to a specific objective, e.g. expected travel time. This problem has not yet been considered in the literature.

Since a strategy under time-adaptive route choice corresponds to a hyperpath in the time-expanded hypergraph $\mathcal{H}$, we have that, finding the K best strategies under time-adaptive route choice, corresponds to finding the K minimum weight hyperpaths in $\mathcal{H}$. Algorithms based on Nielsen, Andersen, and Pretolani [69] are presented in Section 4.1 which are a nontrivial extension of existing algorithms for the problem of finding the K shortest loopless paths in a directed graph. The problem of finding the K shortest loopless paths is a classical problem in graph theory. It was considered as early as 1959 by Hoffman and Pavley [43]. The best algorithm known to date is the algorithm by Yen [94], later discussed by Lawler [51] in the more general framework of finding the K best solutions to a discrete optimization problem.

The problem of finding the K best strategies under a priori route choice corresponds to finding the K minimum weight hyperpaths satisfying that each strategy defines a path in the network. A specialized algorithm based on lower bounds is used to solve this problem in Section 4.2. These results were obtained during the last year of my PhD and will be the subject of a forthcoming paper. Note that, since finding the best strategy under a priori route choice is $\mathcal{NP}$-hard, we have that the problem of finding the K best strategies under a priori route choice is also $\mathcal{NP}$-hard.

Bicriterion route choice in STD networks are considered in Chapter 5, i.e. we assume that two criteria are used e.g. expected travel time and cost. The goal is now to find efficient strategies, i.e. strategies for which it is not possible to find a different strategy such that expected travel time or cost is improved without getting a worse expected cost or travel time, respectively.

In Section 5.4, which is based on Nielsen, Andersen, and Pretolani [70], we consider the problem under time-adaptive route choice. Here the problem consists in finding the set of efficient strategies between an origin and a destination node when leaving the origin at a specific time. To the author's knowledge, no one has yet considered this problem.

Since a strategy corresponds to a hyperpath in the time-expanded hypergraph, finding the set of efficient strategies, corresponds to finding the set of efficient hyperpaths in the time-expanded hypergraph.

Section 5.5 considers the problem under a priori route choice, i.e. we are interested in finding the set of efficient strategies all defining a loopless path in the network. The research presented in Section 5.5 has been done during the last 6 months of my PhD and will be presented in a forthcoming paper. Only one paper has considered this problem, namely Miller-Hooks and Mahmassani [60], with the first objective being minimizing expected time and the second objective being minimizing expected cost. A labelling procedure was presented, which guarantees that all the efficient paths can be obtained; however, in practice the procedure is too slow.

We solve the bicriterion route choice problems under a priori and time-adaptive route choice using a two-phase approach. The two-phase approach is a general method for solving bicriterion combinatorial problems. As the name suggests, the two-phase approach splits the search of efficient strategies into two phases. In phase one, a subset of efficient strategies is found, which defines regions where further efficient strategies may be found. Phase two proceeds to searching these regions one by one using the K best strategies procedures developed in Chapter 4. Unfortunately, the two-phase approach cannot be extended to the case where more than two criteria are considered, since phase one may fail in identifying all regions where further efficient strategies may be found.

Finally, Chapter 6 presents two problems which, due to time issues, have not been studied as deeply as the problems in Chapter 4 and 5. That is, the results given may not be considered as complete but may be regarded as directions for further research. In Section 6.1 we consider the problem of extending the K best strategies and bicriterion algorithms to the case where the leaving time from the origin is not known and in Section 6.2 the problem of finding the best strategy under time-adaptive route choice when the cost of leaving a node at time t along an arc is not known and instead described by a linear function.

1.1 Summary of the thesis

This thesis deals with problems concerning route choice in stochastic time-dependent networks (*STD networks*). STD networks are an extension of more "traditional" networks where the travel time or cost between two nodes (towns, telephone switches etc.) are deterministic and time-independent. In stochastic time-dependent networks the travel time between two nodes is time-dependent, i.e. the travel time depends on the leaving time from a node. Furthermore, it is assumed that for each leaving time, the travel time may not be fully known and hence a probability function is used to express possible travel times.

Route choice problems in STD networks may be regarded as extensions of traditional shortest path problems in directed graphs. The problem of finding a shortest path in a directed graph may be considered as two problems in an STD network, depending on whether the entire route, denoted a *strategy*, must be specified a priori, i.e. before travel begins (*a priori route choice*) or whether the driver is allowed to react while travelling on the revealed/actual arrival times at intermediate nodes (*time-adaptive route choice*).

The problem of finding the best route/strategy under a priori or time-adaptive route choice consists in finding a strategy which is minimal with respect to a specific objective,

e.g. expected travel time.

This thesis focuses on two route choice problems in STD networks, namely the problem of finding the K best strategies under a priori and time-adaptive route choice and bicriterion route choice under a priori and time-adaptive route choice. Here we assume that two criteria are given, e.g. minimizing expected travel time and cost. The goal is now to find efficient strategies, i.e. strategies for which it is not possible to find a different strategy such that expected travel time or cost is improved without getting a worse expected cost or travel time, respectively.

STD networks are modelled using directed hypergraphs which are presented in Chapter 2. Here basic definitions of a directed hypergraph, hyperpath and hypertree is given in Section 2.2. Furthermore, weighted hypergraphs are considered and procedures for finding the minimum weight hyperpath for different weighting functions are presented (Section 2.3). Finally, reoptimization techniques are developed in Section 2.4. These techniques are essential for the efficiency of the algorithms in the thesis. The first sections of the chapter is based on well-known results and hence no proofs are given. The theorems concerning reoptimization techniques in Section 2.4 are new.

Chapter 3 concerns STD networks and is mostly based on well-known results. First, notation and definitions of a strategy under a priori and time-adaptive route choice is given in Section 3.1. In Section 3.2 the different criteria which are used in the thesis are presented. Next, in Section 3.3 it is shown how STD networks can be modelled using hypergraphs and results are presented showing how the problem of finding the best strategy under time-adaptive route choice can be transformed into a minimum weight hyperpath problem. Finally, Section 3.5 presents the TEGP generator developed during my PhD, which is a generator for generating different STD networks.

In Chapter 4, the problem of finding the K best strategies under a priori and time-adaptive route choice is considered. Section 4.1 considers the problem under time-adaptive route choice and is based on Nielsen et al. [69]. It is shown that the problem is equivalent to finding the K minimum weight hyperpaths in a time-expanded hypergraph. A nontrivial extension of the algorithm, presented by Yen [94], for finding the K shortest loopless paths in a directed graph, is used to solve the problem. Furthermore, an improved algorithm using reoptimization is given. Section 4.2 concerns the problem of finding the K best strategies under a priori route choice. The problem can be solved in two ways depending on how we partition the set of strategies. These results were obtained during the last year of my PhD and will be the subject of a forthcoming paper. Finally, computational results under a priori and time-adaptive route choice are presented in Section 4.3 including a short summary of the results.

Chapter 5 concerns the problem of finding the efficient set of strategies under time-adaptive and a priori route choice. In Section 5.4, which is based on Nielsen et al. [70], we consider the problem under time-adaptive route choice while Section 5.5 considers the problem under a priori route choice. The research, presented in Section 5.5, has been done during the last 6 months of my PhD and will be presented in a forthcoming paper.

For both a priori and time-adaptive route choice, the problem is solved using a two-phase approach. The two-phase approach is a general method for solving bicriterion combinatorial problems. As the name suggests, the two-phase method splits the search of efficient strategies into two phases. In phase one a set of efficient strategies is found defining regions where further efficient strategies may be found. Phase two proceeds to searching these regions one by one using the K best strategies procedures developed

in Chapter 4. Computational results under a priori and time-adaptive route choice are presented in Section 5.6 including a short summary of the results.

The last chapter, Chapter 6, presents two problems which, due to time issues, have not been studied as deeply as the problems in Chapter 4 and 5. That is, the results given should not be considered as complete but may be regarded as directions for further research. In Section 6.1 we consider route choice when the leaving time from the origin is not known and in Section 6.2 the problem of finding the best strategy under time-adaptive route choice when the cost of leaving a node at time t along an arc is not known and instead described by a linear function.

Finally, Appendix A considers two problems not directly related to STD networks, namely the subhypertree constrained hyperpath problem and the problem of finding the K minimum weight hyperpaths in a non-acyclic hypergraph. Both emerged as a result of studying the problem of finding the K best strategies in an STD network. Appendix B presents an overview over the basic data structures used in the algorithms. Last, in Appendix C, based on Nielsen and Pretolani [71], we point out that the definition of a hyperpath given by Gallo et al. [29] is wrong. A counter example is presented.

Chapter 2 Directed hypergraphs

Directed hypergraphs are an extension of directed graphs and undirected hypergraphs introduced by Berge [8]. Directed hypergraphs represent a general modelling and algorithmic tool, which have been successfully used in many different research areas such as artificial intelligence, database systems, fuzzy systems, propositional logic and transportation networks. For a more general overview on directed hypergraphs see Ausiello, Franciosa, and Frigioni [3].

The concept of hyperpath and minimum weight hyperpath was introduced by Nguyen and Pallottino [66]. Some particular minimum weight hyperpath problems were studied in Jeroslow, Martin, Rardin, and Wang [46] within the more general setting of Leontief flow problems. The definition of a hyperpath in a directed hypergraph and a general formulation of the minimum weight hyperpath problem were given by Gallo et al. [29].

Minimum weight hyperpath problems have important practical applications, e.g. in production planning (Gallo and Scutellà [32]) and in transportation networks (Pallottino and Scutellà [74]). They are the core in traffic assignment methods for transit networks, see for instance Nguyen and Pallottino [65], Nguyen, Pallottino, and Gendreau [67], Wu, Florian, and Marcotte [93]. For an overview on hyperpath formulations of traffic assignment problems we refer to the textbook by Marcotte and Nguyen [53].

In this chapter we give some basic definitions and properties for a subclass of directed hypergraphs that we use to model stochastic time-dependent networks. The subclass was denoted B-graphs in Gallo et al. [29] which considered the general class of directed hypergraphs. However, as in many papers, the term "hypergraph" is used for the subclass appropriate in the context.

2.1 Basic definitions

A *directed hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = (v_1, ..., v_n)$ is the set of *nodes*, and $\mathcal{E} = (e_1, ..., e_m)$ is the set of *hyperarcs*. A hyperarc $e \in \mathcal{E}$ is a pair $e = (T(e), h(e))$, where $T(e) \subset \mathcal{V}$ denotes the set of *tail* nodes and $h(e) \in \mathcal{V} \setminus T(e)$ denotes the *head* node. Note that a hyperarc has exactly one node in the head, and possibly several nodes in the tail.

The *cardinality* of a hyperarc e is the number of nodes it contains, i.e. $|e| = |T(e)| + 1$.

We call e an *arc* if $|e| = 2$. The *size* of $\mathcal{H}$ is the sum of the cardinalities of its hyperarcs, i.e.

$$size(\mathcal{H}) = \sum_{e \in \mathcal{E}} |e| .$$

Without loss of generality, we assume $size(\mathcal{H}) > n$. We denote by

$$FS(v) = \{e \in \mathcal{E} \mid v \in T(e)\}, \quad BS(v) = \{e \in \mathcal{E} \mid v = h(e)\}$$

the *forward star* and the *backward star* of node v, respectively.

A hypergraph $\tilde{\mathcal{H}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ is a *subhypergraph* of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, if $\tilde{\mathcal{V}} \subseteq \mathcal{V}$ and $\tilde{\mathcal{E}} \subseteq \mathcal{E}$. This is written $\tilde{\mathcal{H}} \subseteq \mathcal{H}$ or we say that $\tilde{\mathcal{H}}$ is *contained* in $\mathcal{H}$. A subhypergraph is *proper* if at least one of the inclusions is strict. Moreover, we denote by $FS_{\tilde{\mathcal{H}}}(v)$ and $BS_{\tilde{\mathcal{H}}}(v)$ the forward and backward star of subhypergraph $\tilde{\mathcal{H}}$ in node v, respectively.

A *valid ordering* $V = (v_1, v_2, \ldots, v_n)$ of $\mathcal{H}$ is a topological ordering of the nodes such that, for any $e \in \mathcal{E}$, if $h(e) = v_i$ and $v_j \in T(e)$ then $j < i$. Note that, in a valid ordering any node $v_j \in T(e)$ precedes node $h(e)$. Let v^{last} denote the *last* node in V, i.e. v_n. A *valid sub-ordering* $\tilde{V}$ of V is a subset of the nodes in V ordered like in V. We write $\tilde{V} \subseteq V$, if $\tilde{V}$ is a valid sub-ordering of V.

2.2 Paths, hyperpaths and hypertrees

A *path* P_{st} in $\mathcal{H}$ is a sequence

$$P_{st} = (s = v_1, e_1, v_2, e_2, ..., e_q, v_{q+1} = t)$$

where, for $i = 1, \ldots, q$, $v_i \in T(e_i)$ and $v_{i+1} = h(e_i)$. A node v is *connected* to node u if a path P_{uv} exists in $\mathcal{H}$. A *cycle* is a path P_{st}, where $t \in T(e_1)$. This is in particular true if $t = s$. If $\mathcal{H}$ contains no cycles, it is *acyclic*.

The next theorem has been proved by Gallo et al. [29], and generalizes a well-known property of acyclic directed graphs.

Theorem 2.2.1 *$\mathcal{H}$ is acyclic if and only if a valid ordering of the nodes in $\mathcal{H}$ is possible.*

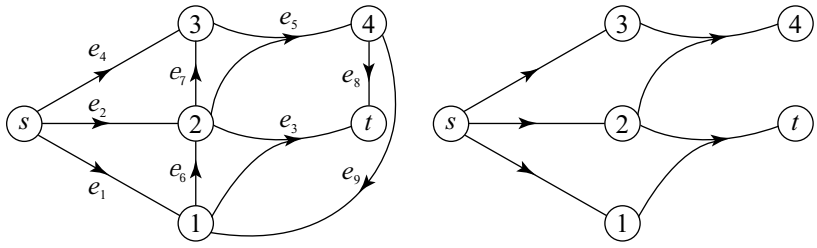
A valid ordering in an acyclic hypergraph is in general not unique, which is also the case for acyclic directed graphs. An $O(size(\mathcal{H}))$ algorithm finding a valid ordering was given by Gallo et al. [29, p195]*.

Different definitions of a hyperpath have been given in the literature. A widely used definition is the one given by Gallo et al. [29]. However, as pointed out by Nielsen and Pretolani [71] this definition is wrong. For the interested reader please see Appendix C. We use the definition given in Ausiello et al. [3].

Definition 2.2.1 A *hyperpath* $\pi_{st} = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ from *source* s to *target* t, is a *subhypergraph* of $\mathcal{H}$ satisfying that, if $t = s$, then $\mathcal{E}_\pi = \varnothing$; otherwise the $q \geq 1$ hyperarcs in $\mathcal{E}_\pi$ can be ordered in a sequence $(e_1, ..., e_q)$ such that

1. $t = h(e_q)$.

*Actually the procedure finds an inverse valid ordering on an F-graph (hyperarcs having only one tail node). However, it can easily be modified.



(a) Hypergraph $\mathcal{H}$ (b) A hypertree in $\mathcal{H}$

Figure 2.1: A non-acyclic hypergraph and a hypertree.

2. $T(e_i) \subseteq \{s\} \cup \{h(e_1), ..., h(e_{i-1})\}, \quad \forall e_i \in \mathcal{E}_\pi$.

3. No proper subhypergraph of π_{st} is an s-t hyperpath.

A node t is *hyperconnected* to s in $\mathcal{H}$ if there exists a hyperpath π_{st} in $\mathcal{H}$. Note that condition 2 implies that a valid ordering of π_{st} is $(s, h(e_1), ..., h(e_q))$. That is, a hyperpath is *acyclic*. Furthermore, condition 3 implies that, for each $u \in \mathcal{V}_\pi \setminus \{s\}$, there exists a unique hyperarc $e \in \mathcal{E}_\pi$, such that $h(e) = u$ and hence for each node $u \in \mathcal{V}_\pi$ there is a unique subhyperpath π_{su} contained in π_{st}. We denote hyperarc e as the *predecessor* of u in π_{st}. The definition of a hyperpath can be extended to *hypertrees*.

Definition 2.2.2 A *directed hypertree* of $\mathcal{H}$ with *root* s is an *acyclic subhypergraph* $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_\mathcal{T})$ with $s \notin \mathcal{N}$ satisfying

$$BS(s) = \emptyset, \quad |BS(v)| = 1, \quad \forall v \in \mathcal{N}$$

It is not difficult to show that a directed hypertree $\mathcal{T}_s$ contains a unique s-u hyperpath for each node $u \in \mathcal{N}$ (see e.g. [29]). That is, $\mathcal{T}_s$ is the union of hyperpaths from s to all nodes in $\mathcal{N}$. Moreover, $\mathcal{T}_s$ can be described by a *predecessor function* $p : \mathcal{N} \to \mathcal{E}$; for each $u \in \mathcal{N}$, $p(u)$ is the unique hyperarc in $\mathcal{T}_s$ which has node u as the head. Note that any hyperpath is a hypertree, in particular, it can be defined by a predecessor function.

Example 1 A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is shown in Figure 2.1(a). A path from 3 to 2 is

$$P_{23} = (3, e_5, 4, e_9, 1, e_6, 2)$$

The path P_{23} is a cycle. The hypergraph $\mathcal{H}$ becomes acyclic when hyperarc e_9 is deleted and has a unique valid ordering, namely $V = (s, 1, 2, 3, 4, t)$. Below we give two hyperpaths in $\mathcal{H}$, namely a hyperpath from s to t and a hyperpath from s to 4.

$$\pi_{st} = \big(\{s, 1, 2, t\}, \{e_1, e_2, e_3\}\big), \quad \pi_{s4} = \big(\{s, 2, 3, 4\}, \{e_2, e_4, e_5\}\big)$$

A hypertree $\mathcal{T}_s$ for all nodes hyperconnected to s in $\mathcal{H}$ is shown in Figure 2.1(b), it is the union of the two hyperpaths given above. Several valid orderings for $\mathcal{T}_s$ exist; one of them is $V = (s, 1, 2, t, 3, 4)$. ■

2.3 Weighted hypergraphs

In this section we introduce weights on hypergraphs, i.e. each hyperarc is assigned a nonnegative real weight. The case where hyperarcs are assigned more than one weight is not considered here, but introduced in Chapter 5.

We consider two different weighting functions, namely the *distance* and the *value*, both of which have been studied in detail (see e.g. Jeroslow et al. [46] and Gallo et al. [29]). Procedures for solving the minimum weight hyperpath problem on non-acyclic and acyclic hypergraphs are presented.

2.3.1 Weighting functions

A *weighted directed hypergraph* is a hypergraph where each hyperarc e is assigned a nonnegative real weight $w(e)$. The weight of a path P_{st} is the sum of the weights of the hyperarcs in P_{st}.

Given an s-t hyperpath π defined by predecessor function p, a weighting function W is a node function assigning weights $W(u)$ to all nodes in π. The weight of hyperpath π is $W(t)$ (or $W(\pi)$). We shall restrict ourselves to *additive weighting functions* introduced by Gallo et al. [29], defined by the recursive equations:

$$W(v) = \begin{cases} 0 & v = s \\ w(p(v)) + F(p(v)) & v \in \mathcal{V}_\pi \setminus \{s\} \end{cases} \tag{2.1}$$

Here $F(e)$ denotes a non-decreasing function of the weights in nodes of $T(e)$. We shall consider two particular weighting functions, namely the *distance* and the *value*. The *distance* function is obtained by defining $F(e)$ as follows:

$$F(e) = \max_{u \in T(e)} \{W(u)\} \tag{2.2}$$

and the *value* function is obtained as follows:

$$F(e) = \sum\nolimits_{u \in T(e)} a_e(u)\, W(u)$$

where $a_e(v)$ is a nonnegative multiplier defined for each hyperarc e and node $v \in T(e)$. With respect to the value function, two interesting cases have been introduced in the literature. If

$$a_e(u) = 1,\ \forall e \in \mathcal{E},\ u \in T(e)$$

then the weighting function is called the *sum* function and if

$$\sum\nolimits_{u \in T(e)} a_e(u) = 1,\ \forall e \in \mathcal{E}$$

then the weighting function is called the *mean* function.

The weight of hyperpath π can be computed using equations (2.1) by processing the nodes forward in a valid ordering V_π of π. However, it may be useful to find the weight of the hyperpath by processing V_π backward, as we will see in the following chapters.

Theorem 2.3.1 *The weight of s-t hyperpath π, using the value weighting function, is*

$$W(\pi) = \sum\nolimits_{u \in \mathcal{V}_\pi \setminus \{s\}} f^\pi(u)\, w(p(u)) \tag{2.3}$$

with f^{π} defined by the following recursive equations

$$f^{\pi}(u) = \begin{cases} 1 & u = t \\ \sum_{e \in FS_{\pi}(u)} a_e(u) f^{\pi}(h(e)) & u \in \mathcal{V}_{\pi} \setminus \{t\} \end{cases} \tag{2.4}$$

The values of f^{π} can be computed by processing the nodes in a valid ordering of π backward.

Proof Let $V_{\pi} = (s = v_1, ..., v_q = t)$ denote a valid ordering of $\pi = (\mathcal{V}_{\pi}, \mathcal{E}_{\pi})$. We show (2.3) by considering each node in V_{π} starting with node v_q processing down to node v_1. Given a set of nodes $I \subseteq \mathcal{V}_{\pi}$, $t \in I$, first, define $f^I(u)$ for all $u \in \mathcal{V}_{\pi}$.

$$f^I(u) = \begin{cases} 1 & u = t \\ 0 & I \cap \{h(e) : e \in FS_{\pi}(u)\} = \emptyset \\ \sum_{\substack{e \in FS_{\pi}(u) \\ h(e) \in I}} a_e(u) f^I(h(e)) & I \cap \{h(e) : e \in FS_{\pi}(u)\} \neq \emptyset \end{cases}$$

Note that $f^I(u)$ is the part of $f^{\pi}(u)$ found by using only the hyperarcs in $FS_{\pi}(u)$ with head belonging to the set I. Moreover, if I is the set of nodes above node u in V_{π} then $f^I(u) = f^{\pi}(u)$. Now consider node v_q. According to equation (2.1), we have that

$$\begin{aligned} W(\pi) &= w(p(v_q)) + \sum\nolimits_{u \in T(p(v_q))} a_{p(v_q)}(u) W(u) \\ &= \sum\nolimits_{v \in I^q} f^{\pi}(v) w(p(v)) + \sum\nolimits_{u \in B^q} f^{I^q}(u) W(u) \end{aligned} \tag{2.5}$$

with

$$I^i = \{v_i, ..., v_q\} \text{ and } B^i = \left\{ u \in \bigcup\nolimits_{v \in I^i} T(p(v)) : u \notin I^i \right\}$$

Assume that nodes $v_q, ..., v_{i+1}$ have been considered and that (2.5) holds, then

$$W(\pi) = \sum\nolimits_{v \in I^{i+1}} f^{\pi}(v) w(p(v)) + \sum\nolimits_{u \in B^{i+1}} f^{I^{i+1}}(u) W(u)$$

It is obvious that $v_i \in B^{i+1}$ and by writing the weight of node v_i out we get

$$\begin{aligned} W(\pi) = \sum\nolimits_{v \in I^{i+1}} f^{\pi}(v) w(p(v)) + \sum\nolimits_{u \in B^{i+1} \setminus \{v_i\}} f^{I^{i+1}}(u) W(u) + \\ f^{I^{i+1}}(v_i) \left(w(p(v_i)) + \sum\nolimits_{u \in T(p(v_i))} a_{p(v_i)}(u) W(u) \right) \end{aligned}$$

Since I^{i+1} consists of all nodes above node v_i in V_{π}, we have that $f^{I^{i+1}}(v_i) = f^{\pi}(v_i)$. Furthermore, note that given node $u \in T(p(v_i))$, we have that if node $u \notin B^{i+1}$, then $f^{I^i}(u)$ becomes $f^{I^i}(u) = f^{\pi}(v_i) a_{p(v_i)}(u)$; otherwise if $u \in B^{i+1}$, then $f^{I^{i+1}}(u) \neq 0$ and$f^{I^i}(u)$ becomes $f^{I^i}(u) = f^{I^{i+1}}(u) + f^{\pi}(v_i) a_{p(v_i)}(u)$. Hence we have the following

$$W(\pi) = \sum\nolimits_{v \in I^i} f^{\pi}(v) w(p(v)) + \sum\nolimits_{u \in B^i} f^{I^i}(u) W(u)$$

Proceeding down to node v_1, we have that (2.5) can be written as

$$W(\pi) = \sum\nolimits_{v \in \mathcal{V}_{\pi} \setminus \{s\}} f^{\pi}(v) w(p(v))$$

since $B^1 = \varnothing$. ■

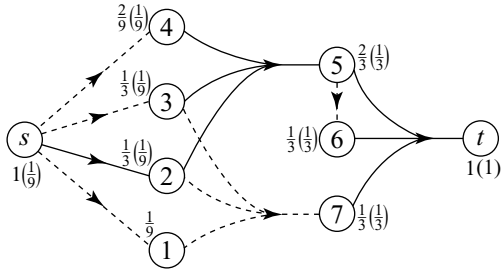


Figure 2.2: A hyperpath π_{st}.

Example 2 A hyperpath π_{st} is shown in Figure 2.2 (the whole figure). A valid ordering of the nodes is $V_\pi = (s, 1, 2, 3, 4, 5, 6, 7, t)$. Assume that the value of the multipliers are equal to $1/|T(e)|$ for each arc e. The value of $f^\pi(v)$ is shown near each node v in Figure 2.2 (the value not in parentheses). The numbers in parentheses will be explained in Section 2.4.1. ■

Note that according to equation (2.2), the weight of a hyperpath using the distance weighting function, is the weight of an maximal weight s-t path contained in π and hence the following theorem holds trivially.

Theorem 2.3.2 *The weight of s-t hyperpath π, using the distance weighting function, is $l^\pi(s)$, where $l^\pi(u)$ denotes the maximum weight of a path from u to t contained in π.*

l^π can be computed by using the following recursive equations

$$l^\pi(u) = \begin{cases} 0 & u = t \\ \max_{e \in FS_\pi(u)} \{l^\pi(h(e)) + w(e)\} & u \in \mathcal{V}_\pi \setminus \{t\} \end{cases} \tag{2.6}$$

and by processing the nodes backwards according to a valid ordering of π.

2.3.2 Minimum weight hyperpaths

The *minimum weight hyperpath problem* or shortest hyperpath problem can be viewed as a natural generalization of the shortest path problem and consists in finding the minimum weight hyperpaths from a source s to all nodes in $\mathcal{H}$ hyperconnected to s. The result is a *minimum weight hypertree* containing minimum weight hyperpaths to all nodes hyperconnected to s.

In general, the problem is $\mathcal{NP}$-hard (see e.g. Ausiello, Nanni, and Italiano [5]), but if the weighting function is additive and cycles are not decreasing, polynomial algorithms exist (see [29]). We define a cycle $C = \{v_1, e_1, v_2, e_2, ..., v_r, e_r, v_{r+1}\}$, $v_{r+1} \in T(e_1)$, to be *decreasing* if the node weight $W(v_{r+1})$ can be decreased through the cycle. A sufficient condition for a cycle not is decreasing in $\mathcal{H}$ is

$$\sum_{i=1}^{r} w(e_i) \geq 0$$

for all cycles in $\mathcal{H}$ provided that the distance function is considered (see Gallo et al. [29]). This is the normal non-negativity condition from standard directed graphs. For the value

```
1  procedure SHT(s, H)
2      for (∀v ∈ V) do W(v) := ∞;
3      for (∀e_j ∈ E) do k_j := 0;
4      Q = {s}; W(s) := 0;
5      while (Q ≠ ∅) do
6          select and remove u ∈ Q such that W(u) = min{W(v) | v ∈ Q};
7          for (e_j ∈ FS(u)) do
8              k_j := k_j + 1;
9              if (k_j = |T(e_j)|) then
10                 v := h(e_j);
11                 if (W(v) > w(e_j) + F(e_j)) then
12                     if (v ∉ Q) then Q := Q ∪ {v};
13                     W(v) := w(e_j) + F(e_j); p(v) := e_j;
14                 end if
15             end if
16         end for
17     end while
18 end procedure
```

Figure 2.3: A procedure for finding the minimum weight hypertree.

function a sufficient condition is

$$\prod_{i=1}^{r} a_{e_i}(v_i) \geq 1 \tag{2.7}$$

which is the gainfree condition from Jeroslow et al. [46].

Now, assume that the weighting function is additive, the weights nonnegative and that no cycles are decreasing. As pointed out in Gallo et al. [29], finding the minimum weight hypertree is equivalent to finding a solution to *Bellman's generalized equations*

$$W(v) = \begin{cases} 0 & v = s \\ \min\limits_{e \in BS(v)} \{w(e) + F(e)\} & v \in \mathcal{V} \backslash \{s\} \end{cases}$$

A general procedure for finding the minimum weight hypertree rooted at s containing the minimum weight hyperpaths was proposed in Gallo et al. [29]. Procedure *SHT*, shown in Figure 2.3, is a particular version which is a simple generalization of Dijkstra's algorithm. Here we always pick a node with minimum weight from the candidate set Q (line 6). In this case, the well-known assumption of nonnegative arc weights in standard directed graphs becomes

$$w(e) + F(T(e)) \geq W(v), \quad \forall v \in T(e),\ e \in \mathcal{E} \tag{2.8}$$

and Dijkstra's theorem can be extended to hypergraphs.

Theorem 2.3.3 *Suppose* (2.8) *holds and* $W(u) = \min\{W(v) \mid v \in Q\}$. *Then* $W(u)$ *is the minimum weight of all hyperpaths from* s *to* u.

As a consequence we have that every node $u \in \mathcal{V}$ is removed from Q at most once. Note that, using Theorem 2.3.3, we have that the order in which we pick the nodes from

```
1 procedure SHTacyclic(s, H)
2     W(v_1) := 0; for (i = 2 to n) do W(v_i) := ∞;
3     for (i = 2 to n) do
4         for (e ∈ BS(v_i)) do
5             if (W(v_i) > w(e) + F(e)) then
6                 W(v_i) := w(e) + F(e); p(v_i) := e;
7         end for
8     end for
9 end procedure
```

Figure 2.4: An acyclic procedure for finding the minimum weight hypertree.

Q, defines a valid ordering of the minimal hypertree. Moreover, if the distance function is considered, then the node u picked form the candidate set on line 6 is the node in $T(e)$, $e \in FS(u)$ with maximal value. Hence we may use $F(e) = W(u)$ instead of $F(e) = \max_{u \in T(e)} \{W(u)\}$ on line 11 and 13.

The complexity of procedure *SHT*, if a heap implementation [see 87] of the candidate set Q is used and (2.8) holds, is $O(m\ log\ n + size(\mathcal{H}))$. If (2.8) does not hold, a node may be reinserted in Q resulting in complexity $O(n \cdot size(\mathcal{H}))$.

If $\mathcal{H}$ is acyclic, a faster procedure exists (see e.g. Gallo and Pallottino [30]). Procedure *SHTacyclic* is shown in Figure 2.4 and needs a valid ordering $V = (s = v_1, ..., v_n)$ of $\mathcal{H}$. Since each hyperarc is examined once, the procedure runs in $O(size(\mathcal{H}))$ time.

Example 1 (continued) Assume that hypergraph $\mathcal{H}$ is assigned weights as shown in Figure 2.5(a) and consider the mean weighting function with the value of the multipliers equal to $1/|T(e)|$ for each arc e. The minimal hypertree is shown in Figure 2.5(b) with the weight $W(v)$ near to each node v. The weight of the minimal s-t hyperpath is 4. The hypertree in Figure 2.5(b) is also minimal with respect to the distance weighting function.

$\mathcal{H}$ does not satisfy the gainfree condition (2.7). That is, condition (2.7) is not a necessary condition. However, no decreasing cycles will be detected by procedure *SHT*. ■

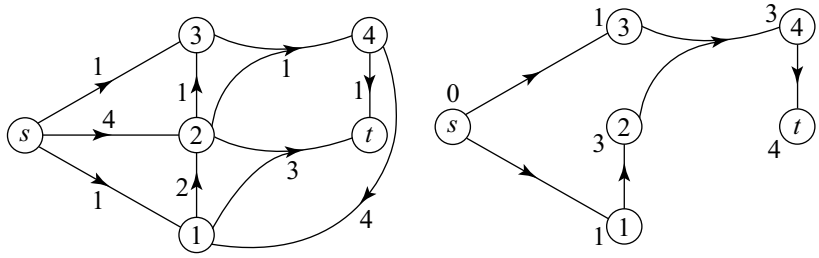
2.4 End-trees and reoptimization techniques

In this section we consider reoptimization techniques on acyclic hypergraphs which later will be used on the acyclic time-expanded hypergraph representing a stochastic time-dependent network.

Let $\mathcal{H}$ be an acyclic hypergraph containing hypertree $\mathcal{T}_s$ defined by predecessor function p. Furthermore, let $W(v)$ denote the weight of the hyperpath $\pi_{sv} \subseteq \mathcal{T}_s$ for each node $v \in \mathcal{N} = \mathcal{V} \setminus \{s\}$.

Definition 2.4.1 Consider an s-t hyperpath π. Then *end-tree* $\eta = (\mathcal{V}_\eta, \mathcal{E}_\eta)$ defined by $I_\eta \subseteq \mathcal{V}_\pi$ and contained in hyperpath π is a hypergraph satisfying

1. $\mathcal{E}_\eta = \bigcup_{v \in I_\eta} p(v)$, $\mathcal{V}_\eta = \bigcup_{e \in \mathcal{E}_\eta} (T(e) \cup \{h(e)\})$



(a) Weights of hypergraph $\mathcal{H}$. (b) The minimum weight hypertree.

Figure 2.5: Hypergraph $\mathcal{H}$ and its minimum weight hypertree.

2. $v \in \mathcal{V}_\eta \Rightarrow \exists v - t$ path in η

Note that, condition 2 implies that $t \in I_\eta$. We denote I_η the set of *inner-nodes* and E_η the set of *leaf-nodes* in η

$$E_\eta = \mathcal{V}_\eta \diagdown I_\eta = \{v \in \mathcal{V}_\eta \mid |BS_\eta(v)| = 0\}$$

Now, assume that each s-t hyperpath in $\mathcal{H}$ must contain end-tree $\eta \subseteq \mathcal{T}_s$, i.e. $BS_\pi(v) = \{p(v)\}$ for all $v \in I_\eta$. Consider nodes $v \in E \subseteq E_\eta$ and assume that subhypergraph $\tilde{\mathcal{H}}$ is obtained by removing the predecessor hyperarc $p(v)$, $\forall v \in E$ from $\mathcal{H}$. Let $\tilde{\mathcal{T}}_s$ be the minimal hypertree of $\tilde{\mathcal{H}}$ and let $\tilde{W}(v)$ denote the weight of the minimal hyperpath $\tilde{\pi}_{sv} \subseteq \tilde{\mathcal{T}}_s$.

We consider the problem of finding the minimal hypertree $\tilde{\mathcal{T}}_s$, in particular the weight of the minimal s-t hyperpath $\tilde{\pi} \subseteq \tilde{\mathcal{T}}_s$. The two weighting functions are considered separately.

2.4.1 The value weighting function

Consider the value weighting function. Using end-tree η, Theorem 2.3.1 can be reformulated.

Theorem 2.4.1 *The weight of hyperpath π, using the value weighting function and end-tree $\eta \subseteq \pi$, is*

$$W(t) = \sum\nolimits_{v \in E_\eta} W(v) f^\eta(v) + \sum\nolimits_{v \in I_\eta} w(p(v)) f^\eta(v) \tag{2.9}$$

where

$$f^\eta(u) = \begin{cases} 1 & u = t \\ \sum_{e \in FS_\eta(u)} a_e(u) f^\eta(h(e)) & u \in \mathcal{V}_\eta \diagdown \{t\} \end{cases} \tag{2.10}$$

The values of f^η can be computed by processing the nodes backwards according to valid ordering $V_\eta \subseteq V_\pi$.

Proof Consider valid ordering $V_\eta = (v_1, ..., v_q = t)$ of η. We show (2.9) by considering each node in V_η starting with node v_q processing down to node v_1. Given $I \subseteq \mathcal{V}_\eta$, $t \in I$ define $f^I(u)$ for all $u \in \mathcal{V}_\eta$

$$f^I(u) = \begin{cases} 1 & I = \{t\} \\ 0 & I \cap \{h(e) : e \in FS_\eta(u)\} = \emptyset \\ \sum_{\substack{e \in FS_\eta(u) \\ h(e) \in I}} a_e(u) f^I(h(e)) & I \cap \{h(e) : e \in FS_\eta(u)\} \neq \emptyset \end{cases}$$

Note $f^I(u)$ is the part of $f^\eta(u)$ found by using only the hyperarcs in $FS_\eta(u)$ with head belonging to the set I. Moreover, if I is the set of nodes above node u in V_η, then $f^I(u) = f^\eta(u)$. Now, consider node v_q. According to equation (2.1) we have that

$$W(t) = w(p(v_q)) + \sum\nolimits_{u \in T(p(v_q))} a_{p(v_q)}(u) W(u) \tag{2.11}$$

Let I_η^i and E_η^i denote the set of inner and leaf-nodes already considered

$$I_\eta^i = \{v_j \in I_\eta : i \leq j \leq q\}, \quad E_\eta^i = \{v_j \in E_\eta : i \leq j \leq q\}$$

Moreover, let

$$B_\eta^i = \left\{ u \in \bigcup\nolimits_{v \in I_\eta^i} T(p(v)) : u \notin I_\eta^i,\ u \notin E_\eta^i \right\}$$

Then (2.11) can be written as

$$\begin{aligned} W(t) = &\sum\nolimits_{v \in I_\eta^q} w(p(v)) f^\eta(v) + \sum\nolimits_{v \in E_\eta^q} f^\eta(v) W(v) \\ &+ \sum\nolimits_{u \in B_\eta^q} f^{I_\eta^q}(u) W(u) \end{aligned} \tag{2.12}$$

Assume that node $v_q, ..., v_{i+1}$ have been considered and that (2.12) holds, then $v_i \in B^{i+1}$. We consider two cases. If $v_i \in I_\eta$, we write out $W(v_i)$, i.e.

$$\begin{aligned} W(t) = &\sum\nolimits_{v \in I_\eta^{i+1}} w(p(v)) f^\eta(v) + \sum\nolimits_{v \in E_\eta^{i+1}} f^\eta(v) W(v) \\ &+ \sum\nolimits_{u \in B_\eta^{i+1} \setminus \{v_i\}} f^{I_\eta^{i+1}}(u) W(u) \\ &+ f^{I_\eta^{i+1}}(v_i) \left(w(p(v_i)) + \sum\nolimits_{u \in T(p(v_i))} a_{p(v_i)}(u) W(u) \right) \\ = &\sum\nolimits_{v \in I_\eta^i} w(p(v)) f^\eta(v) + \sum\nolimits_{v \in E_\eta^i} f^\eta(v) W(v) + \sum\nolimits_{u \in B_\eta^i} f^{I_\eta^i}(u) W(u) \end{aligned}$$

and (2.12) holds for node i. If $v_i \in E_\eta$ then (2.12) also holds since $I_\eta^i = I_\eta^{i+1}$, $E_\eta^i = E_\eta^{i+1} \cup \{v_i\}$ and $B_\eta^i = B_\eta^{i+1} \setminus \{v_i\}$.

Proceeding down to node v_1, we have that (2.12) can be written as

$$W(t) = \sum\nolimits_{v \in E_\eta} f^\eta(v) W(v) + \sum\nolimits_{v \in I_\eta} w(p(v)) f^\eta(v)$$

since $B_\eta^1 = \varnothing$. ■

Assume that weights $\tilde{W}(v)$, $v \in E_\eta$ are known. The increase in $\tilde{W}(t)$ is then

$$\begin{aligned}\tilde{W}(t) - W(t) &= \sum\nolimits_{v\in E_\eta} \tilde{W}(v) f^\eta(v) + \sum\nolimits_{v\in I_\eta} w(p(v)) f^\eta(v) \\ &\quad - \sum\nolimits_{v\in E_\eta} W(v) f^\eta(v) + \sum\nolimits_{v\in I_\eta} w(p(v)) f^\eta(v) \\ &= \sum\nolimits_{v\in E_\eta} \left(\tilde{W}(v) - W(v)\right) f^\eta(v)\end{aligned}$$

and hence the following theorem holds.

Theorem 2.4.2 *The weight of hyperpath $\tilde{\pi} \subseteq \tilde{\mathcal{H}}$ is*

$$\tilde{W}(t) = W(t) + \sum\nolimits_{v\in E_\eta} \left(\tilde{W}(v) - W(v)\right) f^\eta(v) \tag{2.13}$$

If only the last node v^{last} of the valid sub-ordering $V_{E_\eta} \subset V_\pi$ is modified, i.e. $E = \left\{v^{last}\right\}$, then $W(u) = \tilde{W}(u)$, $\forall u \in E_\eta \setminus \left\{v^{last}\right\}$. Moreover, $f^\pi\left(v^{last}\right) = f^\eta\left(v^{last}\right)$ resulting in the following corollary.

Corollary 2.4.1 *The weight of hyperpath $\tilde{\pi} \subseteq \tilde{\mathcal{H}}$ when $E = \left\{v^{last}\right\}$ is*

$$\tilde{W}(t) = W(t) + \left(\tilde{W}\left(v^{last}\right) - W\left(v^{last}\right)\right) f^\pi\left(v^{last}\right) \tag{2.14}$$

Note that, the value of $f^\pi(v)$ and $f^\eta(v)$, $v \in \mathcal{V}_\pi$ is not necessarily equal since v-t paths may exist with hyperarcs $e \notin \mathcal{V}_\eta$.

Example 2 (continued) Consider the hyperpath in Figure 2.2 with valid ordering $V_\pi = (s, 1, 2, 3, 4, 5, 6, 7, t)$. The end-tree defined by $I_\eta = (2, 5, t)$ is shown with solid lines in Figure 2.2. The set of leaf-nodes is $E_\eta = \{s, 3, 4, 6, 7\}$. The value of $f^\eta(v)$ is shown near each node $v \in \mathcal{V}_\eta$ in parentheses. ■

2.4.2 The distance weighting function

Given end-tree η, the maximum weight $l^\eta(u)$ of an u-t path contained in η can be found by using the following recursive equations

$$l^\eta(u) = \begin{cases} 0 & u = t \\ \max_{e\in FS_\eta(u)} \{l^\eta(h(e)) + w(e)\} & u \in \mathcal{V}_\eta \setminus \{t\} \end{cases} \tag{2.15}$$

Similar to the mean case, l^π and l^η are not always equal, since not all u-t paths in π may be contained in η.

Lemma 2.4.1 *Let $\mathcal{P}_\pi$ denote the set of s-t paths in π and let $\mathcal{P}_v \subseteq \mathcal{P}_\pi$, $v \in E_\eta$ be defined by*

$$\mathcal{P}_v = \{P_{sv} \cup P_{vt} \in \mathcal{P}_\pi \mid P_{sv} \subseteq \pi_{sv},\ P_{vt} \subseteq \eta\}$$

That is, $\mathcal{P}_v$ denotes the set of s-t paths consisting of a path in the subhyperpath π_{sv} and a v-t path in η. Then

$$\mathcal{P}_\pi = \bigcup\nolimits_{v\in E_\eta} \mathcal{P}_v \quad \textit{and} \quad \bigcap\nolimits_{v\in E_\eta} \mathcal{P}_v = \varnothing$$

Proof Since each path in $\mathcal{P}_\pi$ must contain at least one leaf-node, we have that $\mathcal{P}_\pi = \bigcup_{v\in E_\eta} \mathcal{P}_v$. Note that each path $P_{vt} \subseteq \eta$, $v \in E_\eta$ contains only one single leaf-node, namely v. Assume that $P \in \mathcal{P}_v$ and $P \in \mathcal{P}_u$, $v \neq u$. Moreover, assume without loss of generality that node v precedes node u in V_π. Then $P_{vt} \subset P$ is a path in η containing node u contradicting that v must be the only leaf-node in P_{vt} and hence $\bigcap_{v\in E_\eta} \mathcal{P}_v = \varnothing$. ∎

Given a node $v \in E_\eta$, we have that $W(v) + l^\eta(v)$ is the maximal weight of an s-t path in $\mathcal{P}_v$. Using Lemma 2.4.1, we have

Theorem 2.4.3 *Given leaf-nodes E_η of η, we have that the weight of π is*

$$W(t) = \max_{v\in E_\eta} \{W(v) + l^\eta(v)\}$$

and the weight of hyperpath $\tilde{\pi}$ is

$$\tilde{W}(t) = \max_{v\in E_\eta} \left\{\tilde{W}(v) + l^\eta(v)\right\} \tag{2.16}$$

If we only modify $\mathcal{H}$ by removing the predecessor from exactly one node $\hat{v}$ (i.e. $E = \{\hat{v}\}$) then some special cases arise

Theorem 2.4.4 *Let P denote a maximal weight path in π.*

$$\text{If } \tilde{W}(\hat{v}) \geq W(\hat{v}) \text{ and } P \in \mathcal{P}_{\hat{v}} \text{ then } \tilde{W}(t) = \tilde{W}(\hat{v}) + l^\eta(\hat{v}) \tag{2.17}$$

$$\text{If } \tilde{W}(\hat{v}) \leq W(\hat{v}) \text{ and } \hat{v} \notin P \text{ then } \tilde{W}(t) = W(t) \tag{2.18}$$

Proof Assume $\tilde{W}(\hat{v}) \geq W(\hat{v})$ and $P \in \mathcal{P}_{\hat{v}}$ then the maximal path $\tilde{P}$ of $\tilde{\pi}$ will be contained in $\tilde{\mathcal{P}}_{\hat{v}}$ and hence $\tilde{W}(t) = \tilde{W}(\hat{v}) + l^\eta(\hat{v})$.

Assume $\tilde{W}(\hat{v}) \leq W(\hat{v})$ and $\hat{v} \notin P$. Since P is still a path in $\tilde{\pi}$ and $\tilde{W}(\hat{v}) + l^\eta(\hat{v}) \leq W(t)$, we have that $\tilde{W}(t) = W(t)$. ∎

If $\hat{v}$ is the last node v^{last} of the valid sub-ordering $V_{E_\eta} \subset V_\pi$ then $W(u) = \tilde{W}(u)$, $\forall u \in E_\eta \setminus \{v^{last}\}$ and $l^\pi(v^{last}) = l^\eta(v^{last})$. Furthermore, if $\tilde{W}(v^{last}) \geq W(v^{last})$ and $P \notin \mathcal{P}_{v^{last}}$ then

$$W(t) = \max_{v\in E_\eta\setminus\{\hat{v}\}} \{W(v) + l^\eta(v)\}$$

and hence (2.17) can be modified to

Corollary 2.4.2 *If $E = \{v^{last}\}$ and $\tilde{W}(v^{last}) \geq W(v^{last})$, then*

$$\tilde{W}(t) = \max\left\{W(t), \tilde{W}(v^{last}) + l^\pi(v^{last})\right\}$$

Note that, if hypertree $\mathcal{T}_s$ is minimal, then we always have that $\tilde{W}(v^{last}) \geq W(v^{last})$.

Chapter 3 Stochastic time-dependent networks

Travel time between an origin and a destination is often the primary objective when routing data, commodities, vehicles etc. in a network. The problem of finding a minimal travel time path, if travel time is deterministic and time-independent, has been the subject of extensive research for many years. For an overview see e.g. Deo and Pang [22] or the textbook by Ahuja, Magnanti, and Orlin [2]. However, a transportation network in which travel times between locations are deterministic and time-independent is often unrealistic. For instance, travel time between one's home and one's workplace is normally faster at midnight than during rush hour, and even during off-peak hours, travel times may vary substantially.

We say that a network is *time-dependent* if the travel times on the arcs are functions of time, and *stochastic* if the travel time is represented by probability distributions rather than simple scalars.

Several papers address stochastic shortest path problems in stochastic time-independent networks. Each with a different meaning of an optimal path. Frank [27] considered the general problem of finding shortest path probability distributions. Sigal, Pritsker, and Solberg [83] introduced the problem where the optimal path is the one with the greatest probability of being the shortest path. The problem generated considerable interest. We point out Adlakha [1] and Corea and Kulkarni [19]. A somewhat related problem considered by Jaillet [45] is the one of determining a path, a priori, that minimizes the expected distance, with some nodes being subject to failure with a known probability. One of the most well considered problems is where the optimal path is the one that maximizes the decision maker's expected utility. The problem was first considered by Loui [52] and generated wide interest, see e.g. Murthy and Sarkar [64] and Eiger, Mirchandani, and Soroush [25]. Recently, the problem of finding a path that minimizes expected length, when the traveller can change his path upon arrival at an intermediate node, has been studied, see e.g. Psaraftis and Tsitsiklis [79] and Provan [78].

Shortest path problems on non-stochastic time-dependent networks have been studied for many years. Nonstochastic time-dependent networks was considered as early as 1966 by Cooke and Halsey [18]. Various types of problems have been considered depending on whether we are considering travel time or cost, discrete vs. continuous representation of time, waiting is allowed in nodes vs. no waiting etc. Orda and Rom [72, 73] considered

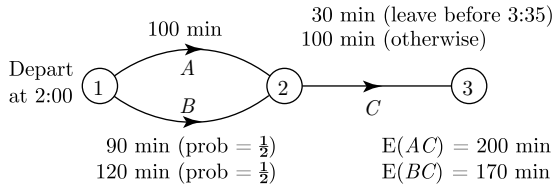


Figure 3.1: Halls example.

continuous time-dependent networks and the problem of finding a path which minimizes the travel time or cost between two nodes when leaving the origin node at time zero. On discrete time-dependent networks, the problem of finding paths that minimize travel time from one origin node, leaving at time zero, to all other nodes has been addressed by Cooke and Halsey [18] and when no waiting is allowed by Kaufman and Smith [50]. The problem of finding paths minimizing travel time from all nodes to one destination node for all leaving times was considered by Ziliaskopoulos and Mahmassani [95] and Chabini [12]. In Chen and Tang [15] the problem of finding a minimal travel time or cost path between two nodes was addressed on a discrete network, where leaving a node is only prohibited at a set of pre-specified leaving times and in Chen, Rinks, and Tang [14] the problem of finding the K minimum cost paths on the same type of network was solved.

It is evident that both the stochastic and time-dependent properties are appropriate in a transportation network model. For instance, the time to travel from one bus stop to another depends on whether or not you arrive at the first stop in time to catch your preferred bus, and regardless of which bus you take, your travel time on the bus will vary. Therefore stochastic time-dependent networks* (*STD networks*) often provide a better modelling tool in transportation applications.

STD networks was first addressed by Hall [37] who considered the problem of finding a path between two nodes minimizing the expected travel time when leaving the origin at a specific time. He pointed out several ways to formulate the path selection problem in STD networks and complications arising as a consequence of modelling both the stochastic and time-dependent properties in the network.

If a loopless path must be specified before travel begins, and no deviations from the route are permitted, the path is selected *a priori* on the basis of only the probability distributions of the arc travel-times. This is referred to as *a priori route choice* and may be useful for routing highly sensitive substances for which the path travelled must be preapproved or where the driver does not have access to (or time to access) information while travelling.

It is well-known that the deterministic shortest path problem can be solved fast since the *principle of optimality* holds. That is, optimal paths are composed of optimal subpaths. Furthermore, as pointed out by Loui [52], the same holds for the problem of finding a path in a stochastic time-independent network with minimum expected travel time. Here the problem can be solved by just using the expected values of the edge probabilities and solve the resulting deterministic problem. Hall [37] presented a simple example showing that the principle of optimality does not hold in STD networks. In

*Also called random time-dependent networks, stochastic time-varying networks or stochastic dynamic networks.

Halls example, reproduced in Figure 3.1, a traveller leaves node 1 at 2 o'clock. Route A has deterministic travel time, route B stochastic travel time and route C time-dependent travel time. Although route A has the lowest expected travel time to node 2, path BC has a lower expected travel time overall than path AC. That is, the optimality principle is violated. Hall [37] proposed an approach combining branch and bound and K shortest paths techniques for determining the minimum expected travel time a priori path. The worst-case complexity of the algorithm is non-polynomial. Moreover, it required the determination of the k'th shortest path having minimum possible arrival time at the destination. No procedure for determining this path was given. Later, Pretolani [77] showed that the problem of finding a minimal expected travel time path under a priori route choice is $\mathcal{NP}$-hard.

Hall [37] also pointed out that rather than to determine a single path based on only information known before travel begins, routes with lower expected travel time may be obtained by allowing the driver to react to revealed (actual) arrival times at intermediate nodes. This is identical to a multistage recourse problem where decisions are taken according to realizations of arc travel time outcomes while travelling. See Birge and Louveaux [9] for a general overview of stochastic recourse models. Hall realized that the best *time-adaptive route* minimizing expected travel time is not necessarily a path but rather an time-adaptive *strategy* that assigns optimal successors to a node as a function of time. This is referred to as *time-adaptive route choice*. He proposed a dynamic solution approach to determine optimal strategies for networks with limited size.

Several papers have emerged from Halls work. Miller-Hooks and Mahmassani [59] presented two procedures for determining the a priori paths with least possible travel time. Such paths may not necessarily be the most desirable, since they do not consider all relevant risk dimensions. Pretolani [77] presented a hypergraph model for STD networks and showed that a time-adaptive strategy corresponds to a hyperpath in a time-expanded hypergraph. Moreover, the best strategy under different criteria, such as minimizing expected or maximum possible travel time or cost, can be found by solving a minimum weight hyperpath problem using different weights and weighting functions. This result is interesting and the basis of this thesis. In Miller-Hooks and Mahmassani [61] a labelling correcting procedure for finding a path with minimum expected travel time under a priori route choice was presented. Furthermore, a labelling correcting procedure for finding a lower bound on the expected travel time was presented. Later the authors realized that the lower bounding procedure with minor changes solves the time-adaptive route problem and extensive computational testing of the problem of finding optimal time-adaptive strategies was conducted in Miller-Hooks [58]. The procedures in Miller-Hooks [58] do not use hypergraph procedures to find optimal strategies. However, optimal strategies are represented by hyperpaths as in Pretolani [77].

3.1 Problem definition

We consider discrete STD networks where departure times are integer, and travel times are independent integer-valued discrete random variables with time-dependent density functions.

Let $G = (N, A)$ be a directed graph with node set N and arc set A. We will refer to G as *the topological network*. Let $o \in N$ and $d \in N$ denote the *origin* and *destination* node in G, respectively. For the sake of simplicity, we assume that G contains no parallel arcs.

Furthermore, no arcs enter node o and no arcs leave node d.

Assume that departure and arrival times belong to a finite *time horizon*, i.e. a set $H = \{0, 1, ..., t_{\max}\}$ of integer values. In practice, we assume that the relevant time period is discretized into time intervals of length δ, that is, the time horizon H corresponds to the set of time instances $0, \delta, 2\delta, ..., t_{\max}\delta$.

For each arc $(u, v) \in A$ let $L(u, v) \subset H$ be the set of possible leaving times from node u along arc (u, v). Moreover, let $L(u)$, $u \neq d$ denote the set of possible leaving times from node u, i.e

$$L(u) = \bigcup_{(u,v) \in FS(u)} L(u, v)$$

and let $L(d)$ denote the set of possible arrival times at node d. For each arc $(u, v) \in A$ and $t \in L(u, v)$, let $X(u, v, t)$ denote the arrival time at node v when leaving node u at time t along arc (u, v). The arrival time $X(u, v, t)$ is a discrete random variable with density

$$\Pr(X(u, v, t) = t_i) = \theta_{uvt}(t_i), \quad t_i \in I(u, v, t)$$

where

$$I(u, v, t) = \{t_1, ..., t_{\kappa(u,v,t)}\}$$

denotes the set of $\kappa(u, v, t)$ possible arrival times at node v when leaving node u at time t along arc (u, v). That is, for each $t_i \in I(u, v, t)$ the probability of arriving at node v at time t_i when leaving node u at time t is $\theta_{uvt}(t_i)$. We assume that travel times are positive, i.e.

$$t_i > t, \quad \forall t_i \in I(u, v, t) \tag{3.1}$$

For the moment we assume that waiting in a node is prohibited. As we will see in Section 3.3 the model can easily be extended to the case where waiting is allowed. We assume the following assumption holds

$$I(u, v, t) \subset L(v), \quad \forall (u, v) \in A,\ t \in L(u, v),\ v \neq d \tag{3.2}$$

That is, the traveller can not get stuck at an intermediate node v. If it is possible to arrive at node v at time t_i then it is also possible to leave v at time t_i. Moreover, since travel times are positive integers and $t_{\max}$ is finite, a traveller arrives at node d within time $t_{\max}$.

Costs can also be considered in the model by letting $c(u, v, t)$, $t \in L(u, v)$ denote the deterministic cost of leaving node u at time t along arc (u, v). Moreover let $g_d(t)$ be the penalty cost of arriving at node d at time t.

Definition 3.1.1 A *strategy* is a function S with domain

$$Dm(S) \subseteq \{(u, t) : u \in N \setminus \{d\},\ t \in L(u)\}$$

assigning to each pair $(u, t) \in Dm(S)$ a successor arc $(u, v) \in FS(u)$. Furthermore, strategy S must satisfy the following conditions

1. If $(u, t) \in Dm(S)$ and $S(u, t) = (u, v) \Rightarrow t \in L(u, v)$.
2. If $(u, t) \in Dm(S)$ and $S(u, t) = (u, v)$, $v \neq d \Rightarrow (v, t') \in Dm(S), \forall t' \in I(u, v, t)$.

Strategy S provides routing choices for travelling from all nodes and leaving times in the domain towards the destination d. That is, a traveller leaving node u at time t travels along arc $S(u,t)$.

A strategy must provide routing choice for all possible arrival times at an intermediate node when following strategy S (condition 2 above). Note that Definition 3.1.1 extends the definition of a strategy given by Pretolani [77] where a strategy was defined for every node $u \in N \setminus \{d\}$ and leaving time $t \in L(u)$. That is,

$$Dm(S) = \{(u,t) : u \in N \setminus \{d\},\ t \in L(u)\} \tag{3.3}$$

and strategy S provides routing choice for travelling from all nodes at all possible leaving times towards the destination node d. However, we may only be interested in strategies providing route choice when leaving a specific node o at a specific time t towards the destination d.

Definition 3.1.2 A strategy S_{ot} providing routing choice for travelling from the origin node o when leaving at time t towards the destination node d is a strategy satisfying

1. $(o,t) \in Dm(S_{ot})$.
2. No pair (u,t') can be removed from $Dm(S_{ot}) \setminus \{(o,t)\}$ such that Definition 3.1.1 still holds.

That is, in strategy S_{ot}, we only consider the nodes and leaving times necessary for defining the route travelled when leaving the origin at time t.

A strategy is a *path-strategy* if we travel along a loopless path in G, i.e. the successor arc is unique and does not depend on time; in other words, a path-strategy must satisfy

$$S(u,t) = S(u,t'), \quad \forall (u,t), (u,t') \in Dm(S) \tag{3.4}$$

Finally, denote by

$$\kappa = \sum_{(u,v)\in A,\ t\in L(u,v)} \kappa(u,v,t)$$

the total number of possible travel times. The value of κ can be considered as the size of the input.

3.2 Route selection criteria

We consider different criteria for comparing strategies introduced by Pretolani [77]. Other criteria for comparing strategies under a priori and time-adaptive route choice have been considered by Miller-Hooks and Mahmassani [62]. However, these criteria seem to be harder to use, since they require a huge number of comparisons between strategies for finding optimal ones.

In the following we consider criteria under time-adaptive route choice and start by considering the most frequently used criterion for ranking optimal strategies, namely expected travel time. Let the random variable $Y_T^S(u,t)$ denote the arrival time at node d when leaving node u at time t following strategy S (index T indicates that arrival time is considered). If we leave node u at time t following arc $S(u,t) = (u,v)$ and arrive at

time t' at node v, then the arrival time at node d is $Y_T^S(v,t')$ and hence, according to e.g. Hoel, Port, and Stone [42, p108], we have that the density of $Y_T^S(u,t)$ is

$$\Pr\left(Y_T^S(u,t)=t_a\right)=\sum_{t'\in I(u,v,t)}\theta_{uvt}(t')\Pr\left(Y_T^S(v,t')=t_a\right) \tag{3.5}$$

where t_a is the arrival time at node d. Furthermore, since $Y_T^S(d,t)=t$ with probability one, the *expected arrival time* $E_T^S(u,t)$ at the destination d can be found by the following recursive equations

$$E_T^S(u,t)=\begin{cases} t & u=d,\ t\in L(d) \\ \sum_{t_i\in I(u,v,t)}\theta_{uvt}(t_i)\,E_T^S(v,t_i) & u\neq d,\ t\in L(u)\end{cases}$$

assuming that $S(u,t)=(u,v)$. The *minimum expected travel time problem* (*MET problem*) consists in finding a strategy S, with domain (3.3), yielding a minimum $E_T^S(u,t)$ for each pair $(u,t)\in Dm(S)$. Note that the expected travel time for a traveller leaving node u at time $t\in L(u)$ is given by $E_T^S(u,t)-t$, that is, expected arrival time and expected travel time only differ by the constant t.

Instead of considering expectation criteria worst cases may be of concern. That is, finding the strategy minimizing maximum possible travel time. Given strategy S, assume that we leave node u at time $t\in L(u)$ following arc $S(u,t)=(u,v)$. Then the *maximum possible arrival time* $M_T^S(u,t)$ at node d is found using the recursive equations

$$M_T^S(u,t)=\begin{cases} t & u=d,\ t\in L(d) \\ \max_{t_i\in I(u,v,t)}\left\{M_T^S(v,t_i)\right\} & u\neq d,\ t\in L(u)\end{cases}$$

The *min-max travel time problem* (*MMT problem*) consists in finding a strategy S, with domain (3.3), giving a minimum $M_T^S(u,t)$ for each pair $(u,t)\in Dm(S)$.

Similarly a random variable denoting the cost of strategy S instead of travel time can be defined. The *expected cost* $E_C^S(u,t)$ when leaving node u at time t following strategy S, can then be found using the recursive equations

$$E_C^S(u,t)=\begin{cases} g_d(t) & u=d,\ t\in L(d) \\ c(u,v,t)+\sum_{t_i\in I(u,v,t)}\theta_{uvt}(t_i)\,E_C^S(v,t_i) & u\neq d,\ t\in L(u)\end{cases}$$

and the *maximum possible cost* $M_C^S(u,t)$ by using the recursive equations

$$M_C^S(u,t)=\begin{cases} g_d(t) & u=d,\ t\in L(d) \\ c(u,v,t)+\max_{t_i\in I(u,v,t)}\left\{M_C^S(v,t_i)\right\} & u\neq d,\ t\in L(u)\end{cases}$$

The *minimum expected cost problem* (*MEC problem*) and the *min-max cost problem* (*MMC problem*) consist in finding a strategy S, with domain (3.3), yielding the minimum of $E_C^S(u,t)$ and of $M_C^S(u,t)$ for each pair $(u,t)\in Dm(S)$, respectively.

The above problems seek a strategy which does not necessarily corresponds to a path, i.e. they are problems under time-adaptive route choice. However, the same problems can be formulated under a priori route choice seeking a path-strategy instead of a "general" strategy denoted the MET, MMT, MEC and MMC problem under a priori route choice.

3.3 A hypergraph model for STD networks

We now use a *time-expanded hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ to model an STD network. The set $\mathcal{V}$ contains one node for each pair (u,t), $t \in L(u)$ and an origin node s, i.e.

$$\mathcal{V} = \left\{u^t : u \in N,\ t \in L(u)\right\} \cup \{s\}$$

For each $(u,v) \in A$ and $t \in L(u,v)$ define a hyperarc

$$e_{uv}(t) = \left(\left\{v^{t_i} : t_i \in I(u,v,t)\right\}, u^t\right)$$

Note that the orientation of $e_{uv}(t)$ is opposite to the orientation of arc (u,v), indeed, the tail of $e_{uv}(t)$ contains a node v^{t_i} for each possible arrival time t_i at node v, when leaving node u at time t. Moreover, for each $t \in L(d)$ define a *dummy arc*

$$e_d(t) = \left(\{s\}, d^t\right)$$

That is, the set of hyperarcs is

$$\mathcal{E} = \{e_{uv}(t) : (u,v) \in A,\ t \in L(u,v)\} \cup \{e_d(t) : t \in L(d)\}$$

It is obvious that $size(\mathcal{H})$ is $O(\kappa)$ and $\mathcal{H}$ can be built in $O(\kappa)$ time. Furthermore, a valid ordering where s precedes the other nodes can be found by ranking the nodes in $\mathcal{V}\setminus\{s\}$ in decreasing order of time since (3.1) holds. Therefore according to Theorem 2.2.1 $\mathcal{H}$ satisfies

Corollary 3.3.1 *$\mathcal{H}$ is an acyclic hypergraph with origin s.*

Observe that there is a one to one correspondence between a strategy and a predecessor function on $\mathcal{H}$, i.e. choosing $p(u^t) = e_{uv}(t)$ is equivalent to choosing $S(u,t) = (u,v)$. Moreover, a node u^t, $u \neq d$ in $\mathcal{H}$ is only defined if $t \in L(u)$ and hence $BS(u^t)$ is nonempty. The same holds for the nodes d^t, $t \in L(d)$ where $e_d(t)$ is the unique arc in $BS(d^t)$. According to Definition 2.2.2, we now have

Corollary 3.3.2 *The predecessor function p defined for each node $u \in \mathcal{V}\setminus\{s\}$ corresponding to strategy S with domain* (3.3) *defines a hypertree $\mathcal{T}_s$ in $\mathcal{H}$. Moreover, a strategy S_{ut} providing route choice when leaving node u at time t towards d defines an s-u^t hyperpath $\pi^{S_{ut}}$.*

Pretolani [77] proved the following important result

Lemma 3.3.1 *The maximum arrival time $M_T^S(u,t)$ for each pair $(u,t) \in Dm(S_{ut})$ using strategy S_{ut} is equal to the weight of hyperpath $\pi^{S_{ut}}$ using the distance weighting function and the following weights*

$$w(e) = \begin{cases} t & e = e_d(t) \\ 0 & otherwise \end{cases} \tag{3.6}$$

The expected arrival time $E_T^S(u,t)$ for each pair $(u,t) \in Dm(S_{ut})$ using strategy S_{ut} is equal to the weight of hyperpath $\pi^{S_{ut}}$ using the mean weighting function, weights (3.6) *and the following multipliers*

$$a_e\left(v^t\right) = \begin{cases} \theta_{uvt}(t_i) & \forall e = e_{uv}(t'),\ t \in I(u,v,t) \\ 1 & \forall e = e_d(t),\ t \in L(d) \end{cases} \tag{3.7}$$

The maximum cost $M_C^S(u,t)$ *for each pair* $(u,t) \in Dm(S_{ut})$ *using strategy* S_{ut} *is equal to the weight of hyperpath* $\pi^{S_{ut}}$ *using the distance weighting function and the following weights*

$$w(e) = \begin{cases} g_t(d) & e = e_d(t) \\ c(u,v,t) & e = e_{uv}(t) \end{cases} \tag{3.8}$$

The expected cost $E_C^S(u,t)$ *for each pair* $(u,t) \in Dm(S_{ut})$ *using strategy* S_{ut} *is equal to the weight of hyperpath* $\pi^{S_{ut}}$ *using the mean weighting function, weights* (3.8) *and multipliers* (3.7).

Lemma 3.3.1 implies that the MET problem, for instance, can be formulated as finding a predecessor function yielding a minimum weight hypertree when the mean weighting function is used. Moreover, since $\mathcal{H}$ is acyclic we have that the MET problem can be solved in $O(size(\mathcal{H})) = O(\kappa)$ time. That is, we have the following theorem for time-adaptive route choice.

Theorem 3.3.1 *Consider the time-expanded hypergraph* $\mathcal{H}$ *of an STD network. Under time-adaptive route choice we have that*

1. *The MET problem can be solved by finding the minimum hypertree using the mean weighting function and weights* (3.6) *and multipliers* (3.7) *on* $\mathcal{H}$.
2. *The MMT problem can be solved by finding the minimum hypertree using the distance weighting function and weights* (3.6) *on* $\mathcal{H}$.
3. *The MEC problem can be solved by finding the minimum hypertree using the mean weighting function and weights* (3.8) *and multipliers* (3.7) *on* $\mathcal{H}$.
4. *The MMC problem can be solved by finding the minimum hypertree using the distance weighting function and weights* (3.8) *on* $\mathcal{H}$.

The above problems can be solved in $O(\kappa)$ *time.*

Unfortunately the same does not hold under a priori route choice where a path-strategy must be found (Pretolani [77]).

Theorem 3.3.2 *The problems in Theorem 3.3.1 under a priori route choice become* $\mathcal{NP}$*-hard.*

The model presented in Section 3.1 assumed that waiting in nodes are not allowed. However, the model can easily be extended. For instance, *waiting* at node u at time t to time $t' = t+1$ can be represented by the arc $e_u(t,t') = \{\{u^{t'}\}, u^t\}$ in $\mathcal{H}$ assigned cost $c(u,t,t')$. Note that assumption (3.2) has to be modified so that, when arriving at a node, the traveller can either wait or leave it. Observe that the size of $\mathcal{H}$ with waiting arcs remains $O(\kappa)$. Other features such as *time windows* can be easily modelled by deleting suitable sets of hyperarcs from $\mathcal{H}$.

Example 3 Consider the topological network $G = (N, A)$ in Figure 3.2, where a is the origin node and d is the destination node. For each arc in G, the possible departure and arrival times are listed in Table 3.1. Here a pair $((u,v),t)$ corresponds to a possible leaving

$(u,v),t$	$(a,b),0$	$(b,c),1$	$(b,c),2$	$(b,d),1$	$(b,d),2$	$(c,d),2$	$(c,d),3$	$(c,d),4$
$I(u,v,t)$	$\{1,2\}$	$\{2,3\}$	$\{3\}$	$\{3\}$	$\{6\}$	$\{3,4\}$	$\{4,5\}$	$\{5,6\}$

Table 3.1: Input parameters.

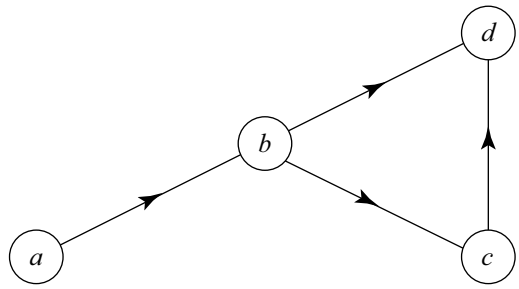


Figure 3.2: The topological network G.

time t from node u along arc (u,v). For the sake of simplicity, we assume that $X\left(u,v,t\right)$ has a uniform density, i.e., for each $t' \in I(u,v,t)$, we have $\theta_{uvt}(t') = 1/|I(i,j,t)|$. For example, if we leave node c at time 2 along arc (c,d), we arrive at node d at time 3 or 4 with the same probability 1/2. Moreover, assume that it is possible to arrive at node c (from node b) at time 2, and leave (towards node d) at time 4. That is, a traveller arriving at node c can *wait* in node c until time 4, and then proceed along arc (c,d); waiting is not allowed elsewhere.

The time expanded hypergraph $\mathcal{H}$ is shown in Figure 3.3. The weight/cost of each hyperarc is given close to each hyperarc. Consider the MMC problem, the minimal hypertree $\mathcal{T}_s$ is shown with solid lines in Figure 3.3. The weight $W\left(u^t\right)$ appears close to each node u^t. Note that $\mathcal{T}_s$ corresponds to a strategy S with sub-strategy S_{a0} corresponding to hyperpath $\pi^{S_{a0}}$. The min-max cost for leaving node a at time zero is $M_C^S(a,0) = 8$. ■

3.4 The density of $Y_T^S\left(o,\hat{t}\right)$

We consider the density of the random variable $Y_T^S\left(o,\hat{t}\right)$ denoting the arrival time at the destination d when leaving the origin o at time $\hat{t}$ following strategy $S_{o\hat{t}}$. The density of $Y_T^S\left(o,\hat{t}\right)$ can be found recursively using (3.5). However, there is a clear connection between f^π, defined in (2.4) and $Y_T^S\left(o,\hat{t}\right)$. Let $Y_T^S\left(o,\hat{t},v\right)$ denote the arrival time at node v when leaving node o at time $\hat{t}$ following strategy $S_{o\hat{t}}$. Then the following holds

Theorem 3.4.1 *Given the hyperpath $\pi^{S_{o\hat{t}}}$ corresponding to strategy $S_{o\hat{t}}$. For each pair $(v,t_a) \in Dm\left(S_{o\hat{t}}\right)$ or pair $\left(d,t_a\right)$, $d^{t_a} \in \mathcal{V}_\pi$, we have that*

$$\Pr\left(Y_T^S\left(o,\hat{t},v\right) = t_a\right) = f^\pi\left(v^{t_a}\right)$$

Proof Consider strategy $S_{o\hat{t}}$ and its corresponding hyperpath $\pi^{S_{o\hat{t}}}$. It is obvious that for pair $\left(o,\hat{t}\right)$, we have that $\Pr\left(Y_T^S\left(o,\hat{t},o\right) = \hat{t}\right) = f(o^{\hat{t}}) = 1$. Moreover, for $v \neq o$, the

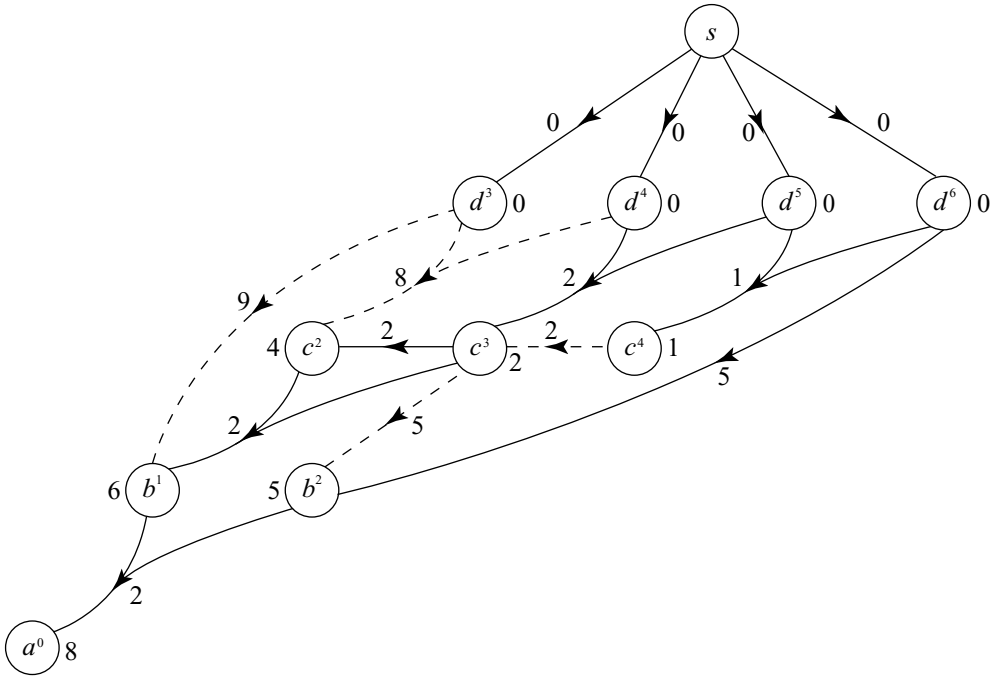


Figure 3.3: The time-expanded hypergraph $\mathcal{H}$.

probability of $Y_T^S\left(o,\hat{t},v\right)$ being equal to t_a, can be defined recursively as

$$\Pr\left(Y_T^S\left(o,\hat{t},v\right)=t_a\right)=\sum_{(u,t):h(S(u,t))=v}\theta_{uvt}\left(t_a\right)\Pr\left(Y_T^S\left(o,\hat{t},u\right)=t\right) \tag{3.9}$$

However, a pair (u,t) satisfying that the head of $h\left(S\left(u,t\right)\right)$ is equal to v corresponds to a hyperarc $e_{uv}\left(t\right)\in FS\left(v^{t_a}\right)$ in π with multiplier $a_e\left(v^{t_a}\right)=\theta_{uvt}\left(t_a\right)$. Therefore the recursive equation (3.9) and the recursive equation (2.4) defining f^π are identical. ■

Given node $v^t\in\mathcal{V}_\pi$, the above theorem says that $f^\pi\left(v^t\right)$ is the probability of arriving at node v at time t, when following strategy $S_{o\hat{t}}$. Since $Y_T^S\left(o,\hat{t},d\right)=Y_T^S\left(o,\hat{t}\right)$, Theorem 3.4.1 gives us that the density of $Y_T^S\left(o,\hat{t}\right)$ is

$$\Pr\left(Y_T^S\left(o,\hat{t}\right)=t_a\right)=f^\pi\left(d^{t_a}\right),\quad d^{t_a}\in\mathcal{V}_\pi$$

and zero otherwise.

3.5 Generating STD networks

To test the algorithms in this thesis, a generator was constructed generating time-expanded hypergraphs, denoted *TEGP* (*Time-Expanded Generator with Peaks*). This program includes several features inspired by typical aspects of road networks (congestion effects,

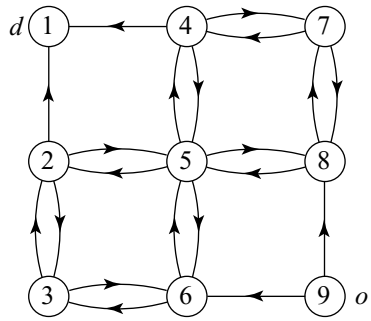


Figure 3.4: A topological grid network (3×3 grid).

waiting, random perturbations etc.). Note that the TEGP generator like other generators only models a fraction of a real network. However, it provides alternative choices that may affect the behavior of the algorithms.

A topological grid network G of *base* b and *height* h is assumed, and we search for optimal strategies from the bottom-right corner node (origin o) to the upper left corner node (destination d). This choice is motivated by the fact that each origin-destination path has at least $b+h-2$ arcs, and there are an exponential number of such paths in G. A topological grid network G of base 3 and height 3 is shown in Figure 3.4.

The generator considers *cyclic* time periods (e.g. a day) and the set of time instances H is the (finite) number of time instances in a cycle multiplied by the number of cycles. In each cyclic period there are some *peak periods* (e.g. rush hours). Each peak consists of three parts; a *transient* part pk_1 where the traffic increases, a *pure peak* part pk_2 where the traffic stays the same and a transient part pk_3 where the traffic decreases again. Peaks are placed at the same time in each cycle. If no peaks is wanted then the length of the peak may be set to zero.

We assume that the travel time density with mean $\mu_{uv}(t)$ and standard deviation $\sigma_{uv}(t)$, for leaving node u at time $t \in H$ along arc (u,v) is a discrete approximation of the normal density. The mean $\mu_{uv}(t)$ follows a pattern like the dotted line in Figure 3.5. In off-peaks $\mu_{uv}(t) = \mu(u,v) \in [lb_T, ub_T]$, where $\mu(u,v)$ denotes the mean travel time in off-peaks. At the beginning of a peak, $\mu_{uv}(t)$ increases from $\mu(u,v)$ to $\mu(u,v)(1+\psi)$, where ψ denotes the *peak increase parameter*, then stays the same during the pure peak period, and then decreases to $\mu(u,v)$ again. The same is the case for the standard deviation, which is defined by $\sigma_{uv}(t) = \rho\mu_{uv}(t)$ where ρ is the *standard deviation mean ratio*. This setting gives higher mean travel time and higher standard deviation in peaks. We assume symmetric mean travel times for the arcs in G, i.e. $\mu_{uv}(t) = \mu_{vu}(t)$.

Two costs $c_i(u,v,t)$, $i = 1,2$ for each arc (u,v) and time $t \in H$ are also generated, since we may need two costs if bicriterion route choice are considered (see Chapter 5). The costs can be generated in two ways, namely using *random costs* or *peak dependent costs*.

If the costs are random no peak effect ($\psi = 0$) are considered and each cost $c_i(u,v,t)$, $i = 1,2$ is generated randomly in $[lb_C, ub_C]$. Note that by using this setting, e.g. the costs of leaving a at time t may be 10 and the costs of leaving a at time $t+1$ may be 500. In a road network this is probably not realistic and peak dependent cost can be generated

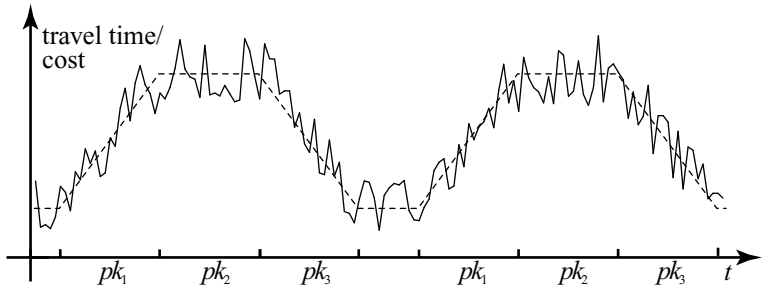


Figure 3.5: Peak effect and random perturbation.

instead. However, random costs can be used to see if the algorithms presented in the following chapters are robust.

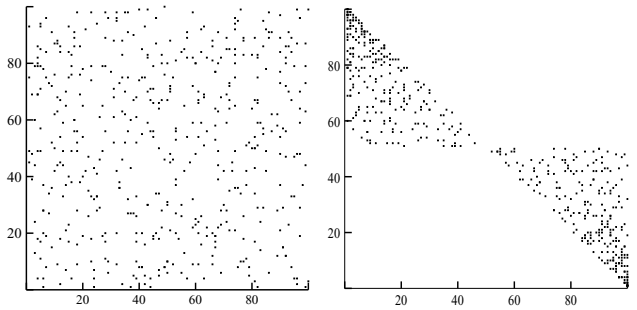
The generation of peak dependent costs takes three components into account: the off-peak cost, the peak effect, and a random perturbation. First, the off-peaks cost $c_i(u,v) \in [lb_C, ub_C]$, $i = 1,2$, are generated. Next, the peak effect for the costs is taken into account. For each arc (u,v), the costs, if leaving node u at a off-peak time, is $\hat{c}_i(u,v,t) = c_i(u,v)$, $i = 1,2$. At the beginning of a peak, the costs $\hat{c}_i(u,v,t)$ increase from $c_i(u,v)$ to $c_i(u,v)(1+\psi)$, then stays the same during the pure peak period, and then decrease to $c_i(u,v)$ again. Finally, the *random perturbation* introduces small variations in the costs, due to other factors not intercepted by the peak implementation, e.g. special information about the cost at exactly that leaving time. For each cost $\hat{c}_i(u,v,t)$, $i = 1,2$, we generate a perturbation $\xi \in [-r_\xi, r_\xi]$, where range r_ξ is a small percentage. Then, the cost $c_i(u,v,t)$ of leaving node u at time t along arc (u,v) becomes $c_i(u,v,t) = \hat{c}_i(u,v,t)(1+\xi)$, $i = 1,2$. Hence the cost follows a pattern like the solid line shown in Figure 3.5. We assume symmetric costs before the random perturbation is applied, i.e. $\hat{c}_i(u,v,t) = \hat{c}_i(v,u,t)$, $i = 1,2$.

If waiting is allowed, waiting costs $c_i(u,t,t+1)$, $i = 1,2$ are generated like for $c_i(u,v,t)$, except that the peak effect is not used for waiting costs. $c_i(u,t,t+1)$ is the cost of waiting in node u from time t to time $t+1$.

The *correlation type* controls the correlation between the costs $c_i(u,v)$, $i = 1,2$. Two correlation types are possible, namely, no_{cor} and neg_{cor}. If the correlation type is no_{cor}, both costs $c_i(u,v)$, $i = 1,2$ are generated randomly in $[lb_C, ub_C]$. If neg_{cor} is used, then the two costs are assumed to be negatively correlated. This is a typical situation in hazardous material transportation, where travel cost and risk/exposure are conflicting. In this case the cost $c_1(u,v)$ is generated in the interval $[lb_C, ub_C]$ and depending on whether the cost is below or above the middle of the interval, the cost $c_2(u,v)$ is generated as follows

$$c_1(u,v) < \frac{ub_C - lb_C}{2} \Rightarrow c_2(u,v) \in [ub_C - (c_1(u,v) - lb_C), ub_C]$$
$$c_1(u,v) \geq \frac{ub_C - lb_C}{2} \Rightarrow c_2(u,v) \in [lb_C, lb_C + (ub_C - c_1(u,v))]$$

Points generated in the interval $[1,100] \times [1,100]$ for the two correlation types are shown in Figure 3.6. Note that if waiting is allowed the waiting costs are correlated according



(a) Cor. type = no_{cor} (b) Cor. type = neg_{cor}

Figure 3.6: Points generated in $[1, 100] \times [1, 100]$ for the different correlation types.

to the specified correlation type.

The time-expanded hypergraph $\mathcal{H}$ is now created by expanding each node $u \in N$ to nodes u^t, $t \in H$. Next, if $t' \leq t_{\max}, \forall t' \in I(u, v, t)$, the hyperarc $e_{uv}(t)$ for each $t \in H$ and $(u, v) \in A$ is created with two weights. The weights depend on the problem considered for criterion i, $i = 1, 2$. If the first criterion is to minimize travel time, the weight $w_1(e_{uv}(t))$ is set to zero (see Lemma 3.3.1), otherwise, if the first criterion is to minimize cost, then the weight $w_1(e_{uv}(t))$ is set to $c_1(u, v, t)$. Similarly for the second criterion. A dummy origin node s and dummy arcs $e_d(t) = (\{s\}, d^t)$ are also created and if the first criterion is to minimize travel time, the weight $w_1(e_d(t))$ is set to t, otherwise a penalty cost may be used (see Lemma 3.3.1). If waiting is allowed, waiting arcs $(\{u_{t+1}\}, u^t)$, $1 \leq t \leq H-1$ for each $u \in N \setminus \{o, d\}$ are generated with weights as described above.

Note that the correlation type and the range r_ξ of the random perturbation do not affect the topological structure of the hypergraph. Moreover, only the interval of possible off-peak travel times $[lb_T, ub_T]$ is given as an input parameter to the TEGP generator. When generating peak dependent costs, the actual interval of possible travel times I_T depends on the peak increase parameter ψ and the standard deviation mean ratio ρ, i.e.

$$I_T = [(1-\rho)\, lb_T, (1+\psi)(1+\rho)\, ub_T] \tag{3.10}$$

The same holds for the generation of costs where also the range of the random perturbation affects the interval of possible costs:

$$I_C = [(1-r_\xi)\, lb_C, (1+\psi)(1+r_\xi)\, ub_C] \tag{3.11}$$

Furthermore, the time horizon H for the STD network corresponding to $\mathcal{H}$ is not given as an input parameter to the TEGP generator. The time horizon depends on the size of G and the possible travel times generated for the arcs in G. The time horizon is set to an upper bound on the travel time needed for travelling from o to d following a path of length $b + h$. Therefore all hyperpaths containing paths of length less than or equal to $b + h$ (not including waiting arcs), will be contained in $\mathcal{H}$. Note that, since H is set to an upper bound, travelling paths of length greater than $b + h$ may often be possible.

Finally, we point out that the generator create nodes and hyperarcs for each $t \in H$. Hence some nodes and hyperarcs may not belong to an source-target hyperpath. These

nodes and hyperarcs can be deleted in a preprocessing step by removing all nodes v^t not hyperconnected to s (including the hyperarcs in $FS(u)$ and $BS(u)$) and all nodes v^t which do not have a path from v^t to the target in $\mathcal{H}$. This results in a hypergraph like the one shown in Figure 3.3, where some of the nodes v^t do not exist.

Chapter 4 Finding the K best strategies in an STD network

One classical problem encountered in the analysis of networks, is the ranking of paths between two nodes in non-decreasing order of length, known as finding the K *shortest paths*. The problem was considered as early as 1959 by Hoffman and Pavley [43].

The applications of the K shortest paths problem are numerous. In Rink, Rodin, and Sundarapandian [80], the problem is applied to routing an aircraft through a network of airfields at an airport and Hadjiconstantinou, Christofides, and Mingozzi [36] use K shortest paths to find lower bounds on vehicle routing problems. In telecommunications, Topkis [88] uses K shortest paths in the context of adaptive routing in communication networks. The problem is also used to compute the K best solutions to general discrete dynamic programming problems (Byers and Waterman [11]) and for solving generalized assignment problems (Chang and Lai [13]). Jiménez and Marzal [47] uses K shortest paths to enumerate the K best sentence hypotheses in speech recognition. Furthermore, practical problems often include constraints which are hard to specify formally or hard to optimize. Here an optimal solution can be found by enumerating suboptimal paths until a path satisfying the hard constraints is found. Last but not least, the K shortest path problem often appears as a subproblem within algorithms for bicriterion shortest path problems, see for example Coutinho-Rodrigues, Climaco, and Current [20] and Handler and Zang [38].

Assume that the K shortest paths problem on a directed network is considered. Usually, two different situations are distinguished. In the *general* (unrestricted) problem, the paths are allowed to be *looping*, i.e. to contain cycles and in the *restricted* problem only *loopless* paths are accepted.

For the general problem, the asymptotically best known algorithm is due to Eppstein [26]. Here the algorithm, after computing the shortest path from every node in the graph to the destination node, builds a graph representing all possible deviations from the shortest path. Once this graph has been built the K shortest paths can be found very fast, i.e. the main time is used on building the graph. A different way to solve the general problem, is to use the path-deletion algorithm by Martins [54] which constructs a sequence of graphs $G_1, ..., G_K$, such that the shortest path in G_k is the k'th shortest path in the graph G. In [6], Azevedo, Costa, Maderira, and Martins avoid the execution of a shortest path algorithm on each graph G_k by properly using information already computed for

G_{k-1}. A further computational improvement was proposed in Azevedo, Madeira, and Martins [7]. More recently, Martins and Santos [57] have proposed a new improvement that reduces the space complexity and in experiments reported by Martins and Santos, their algorithm outperforms in practice the algorithm of Eppstein [26]. In Jiménez and Marzal [48] a new algorithm was proposed. The algorithm recursively computes every new origin-destination path by visiting at most the nodes in the previous path, and using a heap of candidate paths from which the next path is selected. Experimental results show that the new algorithm in practice outperforms the algorithms by Eppstein [26] and Martins and Santos [57]. Recently, also parallel algorithms have been use to solve the general problem, see Guerriero and Musmanno [34] and Ruppert [81].

The restricted problem where only loopless paths are accepted is considered to be harder to solve. The problem was originally examined by Hoffman and Pavley [43], but nearly all early attempts to solve it led to exponential time algorithms, Pollack [76]. The best result known to date is the algorithm by Yen [94], later discussed by Lawler [51] in the more general framework of finding the K best solutions to a discrete optimization problem.

Let $\mathcal{P}$ denote the set of paths between an origin and destination node in a graph $G = (N, A)$ with n nodes and m arcs. In general terms, Yen's algorithm is an implicit enumeration method where $\mathcal{P}$ is partitioned into smaller subsets by recursively applying a branching operation on the current path picked from a candidate set. The algorithm first finds a shortest path P in $\mathcal{P}$,

$$P = (o = u_1, a_1, u_2, a_2, \ldots, a_q, u_{q+1} = d).$$

The branching operation on P then partitions the set $\mathcal{P} \setminus \{P\}$ into q disjoint subsets $\mathcal{P}^i$, $i = 1, ..., q$. Each set $\mathcal{P}^i$ contains the deviations from P at i, that is, each path in $\mathcal{P}^i$ is the concatenation of P_{ou_i} (the subpath of P from o to u_i) and a path from u_i to d not containing arc a_i. It is evident, that by taking the shortest path in $\cup_{i=1,...,q}\mathcal{P}^i$, we find the second shortest path and the branching operation can then be applied on the second minimum weight hyperpath recursively. To find the shortest path in $\cup_{i=1,...,q}\mathcal{P}^i$, we have to find the shortest path in each set $\mathcal{P}^i$. Since each path in $\mathcal{P}^i$ must contain the fixed subpath P_{ou_i}, the shortest path in $\mathcal{P}^i$ can be found by finding a shortest path from u_i to d in a subgraph G^i, obtained from G by deleting each node u_j, $j = 1, ..., i-1$ in P_{ou_i} and deleting arc a_i as well. Yen's algorithm maintains a candidate set representing the partition of $\mathcal{P}$ into subsets. That is, at the beginning of the k'th iteration the first $k-1$ shortest paths have been found, i.e. the candidate set consists of subsets representing the partition of $\mathcal{P} \setminus \{P^1, \ldots, P^{k-1}\}$, where $\{P^1, \ldots, P^{k-1}\}$ are the previously found $k-1$ shortest paths. The k'th shortest path is then selected from the candidate set as the minimal path of all the subsets in the candidate set. The algorithm terminates when the K shortest paths are found, or if the candidate set is empty. Note that Yen's algorithm follows a "forward branching" approach, since we process arcs forward in P, i.e. paths in $\mathcal{P}^i$ must contain path P_{ou_i}, starting at the origin proceeding forward to node u_i. However, we may follow a "backward branching" approach instead, where path P is processed from d towards o. Here each path in the set $\mathcal{P}^i$ would be the concatenation of a path from o to u_{i+1}, not containing arc a_i, and the fixed subpath of P from u_{i+1} to d. From a theoretical point of view, the two approaches are equivalent; as we shall see, this symmetry does no longer hold when we consider finding the K best strategies in an STD network. Since no more than n pairs can be generated at each branching step, Yen's algorithms must solve

at most Kn shortest path problems. Using more recent improvements in shortest path algorithms the complexity of the algorithm becomes $O\left(Kn\left(m + n\log n\right)\right)$.

In the case of undirected graphs, Katoh, Ibaraki, and Mine [49] improved the complexity of Yen's algorithm and in Hadjiconstantinou and Christofides [35] extensive computational experiments was conducted on the algorithm described in [49]. While Yen's asymptotic worst case bound for enumerating the K shortest loopless paths in a directed graph remains unbeaten, several heuristic improvements to his algorithm have been proposed and implemented, as have other algorithms with the same worst case bound. See e.g. Hershberger, Maxel, and Suri [41], Martins and Pascoal [56] and Perko [75].

Similar to what is discussed above for K shortest paths, applications and solution methods for STD networks would take advantage of the availability of alternate optimal or suboptimal solutions. However, to the authors knowledge, no one has considered the problem of finding the K best strategies under a priori or time-adaptive route choice in an STD network.

In this chapter we propose algorithms for finding the K *best strategies* under a priori or time-adaptive route choice between an origin and a destination node leaving the origin at time zero. That is, we consider strategies S_{o0} followed when leaving the origin at time zero and are interested in ranking the first K strategies in non-decreasing order using one of the criteria in Section 3.2.

Since a strategy S_{o0} corresponds to a hyperpath in the time-expanded hypergraph $\mathcal{H}$, finding the K best strategies under time-adaptive route choice corresponds to finding the K *minimum weight hyperpaths* in $\mathcal{H}$ between the source s and the node corresponding to leaving the origin at time zero denoted target t. Note that in our context hyperpaths are acyclic and hence we extend Yen's algorithm for loopless paths to directed hypergraphs; as we shall see, this extension is not straightforward. Moreover, since $\mathcal{H}$ is acyclic in our context, we only consider the problem of finding the K minimum weight hyperpaths in an acyclic hypergraph. The problem, can with some modifications, be extended to non-acyclic hypergraphs. However, this is not relevant for STD networks; therefore the algorithms for the non-acyclic case are given in Appendix A.2.

Similarly finding the K best strategies under a priori route choice corresponds to finding the K minimum weight s-t hyperpaths satisfying that they must correspond to a path-strategy. Unfortunately, due to Theorem 3.3.2, we have that finding the best path-strategy is $\mathcal{NP}$-hard. Therefore the problem of finding the K best path-strategies is also $\mathcal{NP}$-hard. A specialized algorithm based on lower bounds is used to solve this problem in Section 4.2.

Before we start considering STD networks, other areas where K minimum weight hyperpaths algorithms may be used, deserves to be mentioned. *Propositional satisfiability problems* represent a research area where directed hypergraphs is used from a theoretical as well as a practical point of view, see e.g. Gallo, Gentile, Pretolani, and Rago [28]. For instance, the maximum satisfiability problem for Horn formulas (Max Horn SAT) turns out to be equivalent to the problem of finding a minimum cut in a hypergraph (see [28]). Since Max Horn SAT is $\mathcal{NP}$-hard, so is finding the minimum cut in a hypergraph, opposed to the well-known minimum cut problem in graphs. A branch and cut algorithm for finding a minimum cut has been devised by Gallo et al. [28]. This algorithm is based on a particular cut generation technique that requires finding hyperpaths with a weight less than one, using the *cost* weighting function. Here the weight of a hyperpath is the sum of the weights of its hyperarcs. It has been proved by Ausiello, Italiano, and

Nanni [4] that the problem of finding a minimum cost s-t hyperpath is strongly $\mathcal{NP}$-hard. Therefore, finding hyperpaths with a cost less than one is in general hard. In order to overcome this difficulty, a quite simple heuristic is adopted in [28]. This heuristic consists in computing minimum weight hyperpaths for the sum and distance weighting functions which are additive weighting functions, i.e. a minimum weight hyperpath can be found in polynomial time. Even though this approximation performs well for some classes of instances, more sophisticated techniques seems to be necessary to improve the effectiveness of the algorithm. To this aim, K minimum weight hyperpaths procedures can be used. Since the sum weight gives an upper bound on the cost weight, one may try to find all hyperpaths with sum weight less than or slightly larger than one. This defines a (possibly empty) set of cuts that can be generated "easily". Furthermore, the distance weight gives a lower bound on the cost weight. Hence finding all hyperpaths with a distance weight less than one, defines a superset of the valid cuts.

Besides the specific application to cut generation discussed above, it is apparent that a K minimum weight hyperpaths procedure can be used to find a minimum weight s-t hyperpath when the cost weighting function is considered. More precisely, we rank hyperpaths by using the distance weight, keeping track of the minimum cost hyperpath generated in the process. The procedure terminates as soon as the distance weight of the next ranked hyperpath is greater than or equal to the minimum cost weight found so far. Here we exploit the fact that the distance weight is a lower bound on the weight when using the cost weighting function. The same approach can be used to solve a minimum weight hyperpath problem for any "difficult" weighting function, besides cost.

Hypergraphs have been used for many years to model *transit networks* proposed by Nguyen and Pallottino [65]. Transit networks consist of a set of bus lines connected to stop nodes where passengers board or unboard buses. In the hypergraph model, a hyperarc represents the set of attractive bus lines for a passenger waiting at a stop node; a minimum weight hyperpath represents a set of attractive origin-destination routes. The hypergraph model is embedded within a traffic assignment model, based on Wardrop's equilibrium, where passengers are assumed to travel along their minimum weight available hyperpaths. In the context of iterative methods for traffic assignment, it may be computationally useful to identify alternate optimal hyperpaths by using a K minimum weight hyperpaths procedure.

Finally, algorithms for finding the K minimum weight hyperpaths may also be used to solve *minimum makespan assembly problems*. As shown by Gallo and Scutellà [31], an assembly line can be represented by a suitable hypergraph, where each hyperarc represents a machine operation linking two or more subassemblies together. A hyperpath thus represents a particular assembly plan. Assuming that each operation has a cost as well as an execution time, minimum weight hyperpaths with respect to two particular weighting functions give assembly plans with minimum total cost or minimum execution time (with an unlimited number of machines), respectively. Observe that a "good" assembly plan should represent a trade-off between execution time and cost; clearly, this is related to finding efficient hyperpaths which will be discussed in Chapter 5. In general, scheduling a given assembly plan on a fixed number of machines in order to minimize its makespan is a hard problem; approximating methods are discussed in [31]. A possible approach for refining these methods would be to generate K candidate assembly plans; an "optimal" plan would then be chosen according to its approximated minimum makespan scheduling, possibly taking into account other objectives.

4.1 Finding the K best strategies under time-adaptive route choice

Consider an STD network with topological network G and time-expanded hypergraph $\mathcal{H}$. Assume that one of the criteria in Section 3.2 is considered, i.e. $\mathcal{H}$ is assigned weights and weighting function according to Lemma 3.3.1. Furthermore, let

$$V_{\mathcal{H}} = (s = v_1, ..., v_n = t)$$

denote a valid ordering of the nodes in $\mathcal{H}$. We consider the problem of finding the K *best strategies* under time-adaptive route choice between an *origin* node o and a *destination* node d in G when leaving the origin at time zero. That is, we are interested in ranking the first K strategies S_{o0} in non-decreasing order using one of the criteria in Section 3.2. According to Corollary 3.3.2 a strategy S_{o0} corresponds to a hyperpath in $\mathcal{H}$. Hence finding the K best strategies under time-adaptive route choice, corresponds to finding the K *minimum weight hyperpaths* in $\mathcal{H}$ between the *source* s and the node in $\mathcal{H}$ corresponding to leaving node o at time zero, denoted *target* t. That is, we have to extend Yen's algorithm to finding the K minimum weight s-t hyperpaths in $\mathcal{H}$.

Let Π^1 denote the set of s-t hyperpaths in $\mathcal{H}$. In order to extend Yen's algorithm to hypergraphs, we need to devise a suitable branching operation, i.e. a way to partition Π^1 into subsets so that the minimum weight hyperpath in each subset can be found with procedure *SHTacyclic*. Assume that a minimum weight s-t hyperpath π^1 in Π^1 is known (the first minimum weight hyperpath), and defined by predecessor function p^1. Moreover, a valid ordering $V_{\pi^1} \subseteq V_{\mathcal{H}}$ of π^1 is given

$$V_{\pi^1} = (s, u_1, u_2, \ldots, u_{q_1} = t)$$

Since a path is much simpler than a hyperpath, a direct application of Yen's method is not possible. However, since a valid ordering of the nodes in π^1 are given, we may follow a forward branching approach, where we fix the predecessor hyperarcs of the nodes in V_{π^1} forward.

Branching Operation 4.1.1 *Given minimum weight hyperpath π^1 of Π^1 and valid ordering V_{π^1}, the set $\Pi^1 \setminus \{\pi^1\}$ can be partitioned into q_1 disjoint subsets $\Pi^{1,i}$, $1 \leq i \leq q_1$ as follows*

1. *Hyperpaths in $\Pi^{1,1}$ do not contain hyperarc $p^1(u_1)$.*
2. *For $1 < i \leq q_1$, hyperpaths in $\Pi^{1,i}$ contain hyperarcs $p^1(u_j)$, $1 \leq j \leq i-1$, and do not contain hyperarc $p^1(u_i)$.*

Proof Note that hyperpaths in $\Pi^{1,i}$ must contain hyperarcs $p^1(u_1), ..., p^1(u_{i-1})$ and not hyperarc $p^1(u_i)$. Hence the partition of Π^1 defines a branching tree as shown in Figure 4.1 where an arc "fix" on level i means that an s-t hyperpath must contain hyperarc $p^1(u_i)$ and an arc "remove" on level i states that an s-t hyperpath must not contain hyperarc $p^1(u_i)$. It is obvious that the sets $\Pi^{1,i}$, $1 \leq i \leq q_1$ are disjoint and that π^1 is defined by the leaf of the branching tree where all hyperarcs in π^1 have been fixed. Therefore $\cup_{i=1,...,q_1} \Pi^{1,i} = \Pi^1 \setminus \{\pi^1\}$. ■

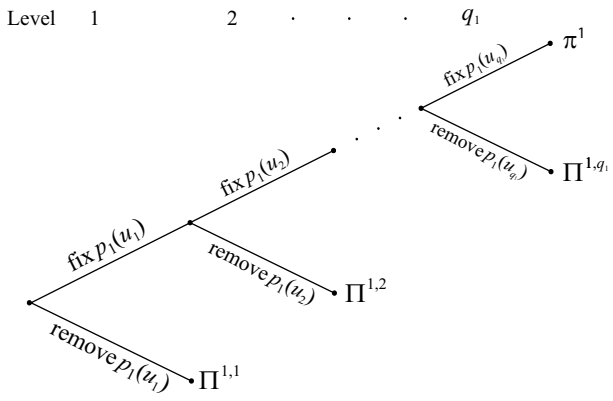


Figure 4.1: The branching tree partitioning Π^1.

A hyperpath in $\Pi^{1,i}$ must contain hypergraph $\mathcal{T}_s^i = (\{s\} \cup \mathcal{N}, \mathcal{E}_\mathcal{T})$ where $\mathcal{N} = \mathcal{E}_\mathcal{T} = \emptyset$ for $i = 1$, and $\mathcal{N} = \{u_1, ..., u_{i-1}\}$ and $\mathcal{E}_\mathcal{T} = \left\{p^1(u_1), ..., p^1(u_{i-1})\right\}$ for $i > 1$. According to Definition 2.2.2, $\mathcal{T}_s^i$ is a hypertree; $\mathcal{T}_s^i \subset \pi^1$ is the hypertree spanning the first $i-1$ nodes of π^1.

Now, consider the problem of finding the minimum weight hyperpath in $\Pi^{1,i}$, $i = 1, ..., q_1$ when using Branching Operation 4.1.1, i.e. a minimum weight hyperpath $\pi^{1,i}$ containing hypertree $\mathcal{T}_s^i$. Note that the problem is not equivalent to finding a minimal hypertree $\mathcal{T}_s$, spanning all nodes in $\mathcal{H}$, containing $\mathcal{T}_s^i$ since the s-t hyperpath π contained in $\mathcal{T}_s$ may satisfy $\mathcal{T}_s^i \nsubseteq \pi$. In fact finding $\pi^{1,i}$ turns out to be difficult.

Theorem 4.1.1 *The problem of finding the minimum weight hyperpath $\pi^{1,i} \in \Pi^{1,i}$ using Branching Operation 4.1.1 is $\mathcal{NP}$-hard.*

Proof The proof in a more general framework is given in Appendix A.1. ∎

4.1.1 Using a backward branching approach

Due to Theorem 4.1.1 we shall follow a different approach, in particular one based on backward branching where we fix the predecessor hyperarcs by processing the nodes in V_{π^1} backward. As we will see, backward branching partitions the set Π^1 in a way that simplifies finding the minimum weight hyperpath in each subset. This result essentially comes from the fact that there is only one node in the head of each hyperarc. The following branching operation is defined in a more general setting than Branching Operation 4.1.1, since it is applied recursively on the first, second,..., K-1'th hyperpath. Let the k'th hyperpath π^k with valid ordering

$$V_{\pi^k} = (s, u_1, ..., u_{q_k} = t)$$

be defined by predecessor function p^k. Assume that π^k is picked from the set Π^k of s-t hyperpaths.

Branching Operation 4.1.2 *Given the k'th minimum weight hyperpath π^k of Π^k and valid ordering V_{π^k}, the set $\Pi^k \setminus \{\pi^k\}$ can be partitioned into q_k disjoint subsets $\Pi^{k,i}$, $1 \leq i \leq q_k$ as follows*

1. *Hyperpaths in Π^{k,p_k} do not contain hyperarc $p^k(u_{q_k})$, that is $p^k(t)$.*
2. *For $1 \leq i < q_k$, hyperpaths in $\Pi^{k,i}$ contain hyperarcs $p^k(u_j)$, $i+1 \leq j \leq q_k$, and do not contain hyperarc $p^k(u_i)$.*

Proof Note that hyperpaths in subset $\Pi^{k,i}$ must contain hyperarcs $p^k(u_{q_k}), ..., p^k(u_{i+1})$ and not hyperarc $p^k(u_i)$, i.e. like in the proof of Branching Operation 4.1.1, the partition of Π^k defines a branching tree as shown in Figure 4.1 ($p^k(u_i)$ is replaced with $p^k(u_{q_k+1-i})$). Hence the sets $\Pi^{k,i}$, $1 \leq i \leq q_k$ are disjoint and we have that $\cup_{i=1,...,q_k} \Pi^{k,i} = \Pi^k \setminus \{\pi^k\}$. ■

Note that, if $k = 1$ in Branching Operation 4.1.2, then the second minimum weight hyperpath can be found by taking the minimum weight hyperpath in the sets $\Pi^{1,i}$, $i = 1, ..., q_1$. However, it is obvious that we can use Branching Operation 4.1.2 recursively.

Definition 4.1.1 For each $i = 1, ..., q_k$, let hypergraph $\eta^{k,i} = (\mathcal{V}_\eta, \mathcal{E}_\eta)$ be defined by

$$\mathcal{E}_\eta = \begin{cases} \emptyset & i = q_k \\ \left\{p^k(u_{q_k}), ..., p^k(u_{i+1})\right\} & \text{otherwise} \end{cases}$$

$$\mathcal{V}_\eta = \begin{cases} \{t\} & i = q_k \\ \bigcup_{e \in \mathcal{E}_\eta} T(e) \cup h(e) & \text{otherwise} \end{cases}$$

Since we fix hyperarcs backward in V_{π^k}, a hyperpath in $\Pi^{k,i}$ must contain $\eta^{k,i}$. According to Definition 2.4.1, $\eta^{k,i}$, $i = 1, ..., q_k - 1$ is an end-tree defined by the nodes $I_\eta = \{u_{i+1}, ..., u_{q_k}\}$*. Now consider the problem of finding the minimum weight hyperpath $\pi^{k,i} \in \Pi^{k,i}$ using Branching Operation 4.1.2. That is, finding a minimum weight s-t hyperpath containing end-tree $\eta^{k,i}$. In this case, the problem reduces to solving a minimum weight hypertree problem on a subhypergraph of $\mathcal{H}$.

In the following we present a general definition on how subhypergraphs can be created when using Branching Operation 4.1.2 on the k'th hyperpath π^k rather than only considering the first minimum weight hyperpath π^1. The hypergraph $\mathcal{H}^k$ is a hypergraph where the k'th hyperpath π^k is minimal and created by using Definition 4.1.2 on one of the $k-1$ hyperpaths $\pi^1, ..., \pi^{k-1}$ previously found by the procedure. If $k = 1$ then $\mathcal{H}^1 = \mathcal{H}$.

Definition 4.1.2 Given π^k, let subhypergraph $\mathcal{H}^{k,i}$, $i = 1, ..., q_k$ be obtained from $\mathcal{H}^k$ as follows

1. For each node u_j, $i+1 \leq j \leq q_k$, remove each hyperarc in $BS(u_j)$ except $p^k(u_j)$.
2. Remove hyperarc $p^k(u_i)$ from $BS(u_i)$.

*The hypergraph $\eta^{k,q_k} = (\{t\}, \emptyset)$ is not an end-tree. But may be considered as one with $E_\eta = \{t\}$ and $I_\eta = \emptyset$.

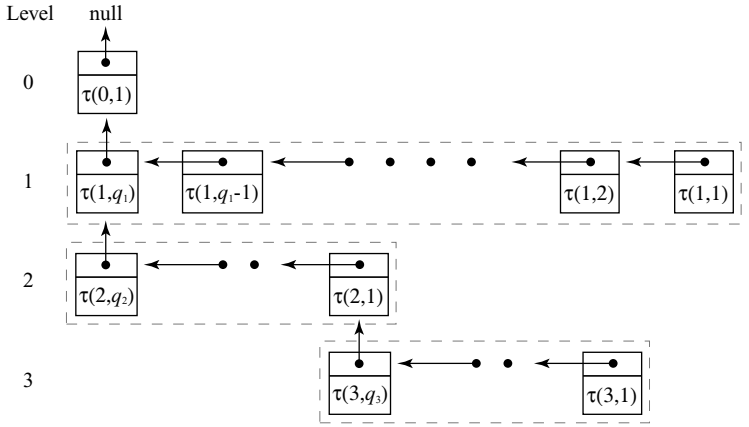


Figure 4.2: The branching tree representing the partition of Π.

Theorem 4.1.2 *Given subhypergraph $\mathcal{H}^{k,i}$, the following are equivalent for $i = 1, ..., q_k$*

1. *$\pi \in \Pi^{k,i}$.*
2. *π is an s-t hyperpath in $\mathcal{H}^{k,i}$.*

Proof Consider $\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi) \in \Pi^{k,i}$ defined by predecessor p. Note that $p(u)$, $u \in \mathcal{V}_\pi$, is a hyperarc in $\mathcal{H}^{k,i}$ and hence we have that π is a hyperpath in $\mathcal{H}^{k,i}$. Next, assume that π is a hyperpath in $\mathcal{H}^{k,i}$. Since all hyperarcs except $p^k(u_j)$ have been removed from $BS(u_j)$, $j = i+1, ..., q_k$, we have that π must contain $\eta^{k,i}$. Furthermore, hyperarc $p^k(u_i)$ has been removed from $\mathcal{H}^{k,i}$, i.e. π cannot contain $p^k(u_i)$ and hence must $\pi \in \Pi^{k,i}$. ■

Note that, in hypergraph $\mathcal{H}^{k,i}$ we remove all hyperarcs in $BS(u_j)$, $i+1 \leq j \leq q_k$, except $p^k(u_j)$, i.e. we fix the backward star of node u_j to $p^k(u_j)$. We say that $\mathcal{H}^{k,i}$ is obtained from $\mathcal{H}^k$ by *fixing* hyperarcs $p^k(u_j)$, $i+1 \leq j \leq q_k$ and *removing* hyperarc $p^k(u_i)$. Theorem 4.1.2 provides us with the following useful result.

Corollary 4.1.1 *Finding the minimum weight hyperpath $\pi^{k,i} \in \Pi^{k,i}$, $i = 1, ..., q_k$, when using Branching Operation 4.1.2, reduces to solving a minimum weight hyperpath problem on $\mathcal{H}^{k,i}$.*

As a consequence, each set $\Pi^{k,i}$ can be represented by its corresponding subhypergraph $\mathcal{H}^{k,i}$.

In order to find the K minimum weight hyperpaths, we implicitly have to maintain a candidate set of pairs $(\tilde{\pi}, \tilde{\mathcal{H}})$, where $\tilde{\pi}$ is a minimum weight hyperpath in $\tilde{\mathcal{H}}$. Assuming that the first k minimum weight hyperpaths $\pi^1, ..., \pi^k$ have been found, the candidate set represents a partition of $\Pi^1 \setminus \{\pi^1, ..., \pi^k\}$. Hyperpath π^{k+1} is then found by picking and removing the pair containing the hyperpath with minimum weight in the candidate set.

Note that, when branching on π^k, hypergraph $\mathcal{H}^{k,i+1}$ only differs slightly from $\mathcal{H}^{k,i}$. In both hypergraphs, hyperarcs $p^k(u_{q_k}), ..., p^k(u_{i+2})$ are fixed; in $\mathcal{H}^{k,i+1}$ we remove hyperarc $p^k(u_{i+1})$ and in $\mathcal{H}^{k,i}$ we fix $p^k(u_{i+1})$ and remove $p^k(u_i)$. Hence $\mathcal{H}^{k,i}$ can be obtained

```
1 procedure setF(τ)
2     remove e_τ from H;
3     while (e_τ ≠ null) do
4         τ := point(τ);
5         if (level changed) then remove e_τ from H;
6         else fix e_τ in H;
7     end while
8 end procedure
```

Figure 4.3: Subprocedure *setF* of procedure *K-BS*.

by first fixing $p^k(u_{i+1})$ and next removing $p^k(u_i)$ from $\mathcal{H}^{k,i+1}$. As a result we can store each pair $\left(\pi^{k,i}, \mathcal{H}^{k,i}\right)$ implicit in a branching tree as shown in Figure 4.2 (the three first minimum weight hyperpaths shown). The branching tree satisfies

1. For $k = 1, 2, 3$ and $i = 1, ..., q_k$, each branching tree node $\tau(k,i) = \left(p^k(u_i), W^{k,i}\right)$, where p^k is the predecessor function defining π^k and $W^{k,i}$ is the weight of a minimum weight hyperpath in $\mathcal{H}^{k,i}$. $\tau(k,i)$ represents pair $\left(\pi^{k,i}, \mathcal{H}^{k,i}\right)$. We assume that the root node $\tau(0,1) = \left(null, W\left(\pi^1\right)\right)$.

2. The arcs represent a pointer to the next branching tree node.

3. All nodes in a dotted square represent a partition of hypergraph $\mathcal{H}^k$ into subhypergraphs $\mathcal{H}^{k,i}$, where $\mathcal{H}^k$ is the hypergraph defined by the branching tree node in the previous level of the branching tree pointed to by the first branching tree node in the square.

4. Vertical pointers indicate that a branching operation on the hypergraph the pointer points to has been performed.

A heap† of pointers to the branching tree nodes currently in the candidate set can be used to order the nodes in the branching tree. The heap data structure is well-known (see e.g. Tarjan [87]) and has a $O(\log Kn)$ worst time complexity for inserting or deleting a branching tree node.

Note there is only one pointer out of each branching tree node and hence there is a unique path between each node τ and the root of the branching tree. Therefore subhypergraph $\mathcal{H}^{k,i}$ can be built by traversing the path from $\tau(k,i)$ to the root node in the branching tree (see procedure *setF* below). A procedure *K-BS* for finding the K best strategies, i.e. the K minimum weight hyperpaths can now be formulated. A backward and forward representation of the time-expanded hypergraph $\mathcal{H}$ and the topological network G is used. For further details on data structures representing and linking $\mathcal{H}$ and G, see Appendix B. The following subprocedures are used.

setF(τ): Builds the subhypergraph corresponding to branching tree node $\tau = (e_\tau, W_\tau)$, see Figure 4.3. Here *point*(τ) denotes the branching tree node which

†In the current implementation a 4-heap is used.

```
1 procedure findOrd(π)
2     inPath(t) = true;
3     for (i = n to 2) do
4         if (inPath(v_i)) then
5             OUTPUT v_i to V_π;
6             for (u ∈ T(p(v_i))) do inPath(u) = true;
7         end if
8     end for
9 end procedure
```

Figure 4.4: Subprocedure *findOrd* of procedure *K-BS*.

τ points at. It is obvious that a hyperarc can be removed from $\mathcal{H}$ by maintaining a flag for each hyperarc equal to zero if not in the subhypergraph and one otherwise. A hyperarc e is then fixed by setting all the flags of the hyperarcs in $BS(h(e))$ equal to zero except the flag for hyperarc e. Since the procedure only removes a hyperarc once, worst case complexity becomes $O(m)$.

SHTacyclic$(s, \mathcal{H})$: Find the minimum weight hypertree of $\mathcal{H}$, i.e. node labels $W(u)$ and $p(u), \forall u \in \mathcal{V}$ (see Figure 2.4). Procedure *SHTacyclic* runs in $O(size(\mathcal{H}))$ time (see Section 2.3.2).

modF$(\tilde{\mathcal{H}}, e_r, e_f)$: Modify the flags in $\mathcal{H}$ by removing hyperarc e_r in $\tilde{\mathcal{H}}$ and fixing the hyperarc e_f (if e_f not equal to *null*).

delMin(): Pick and remove the pointer to the branching tree node $\tau = (e_\tau, W_\tau)$ with minimal label W_τ in the heap. Takes $O(\log Kn)$ time (see Tarjan [87]).

insert(τ): Insert $\tau = (e_\tau, W_\tau)$ into the branching tree. Moreover, if $W_\tau < \infty$, insert a pointer to τ into the heap. It takes $O(1)$ time to add τ to the branching tree and $O(\log Kn)$ time to insert it into the heap, i.e. procedure *insert* runs in $O(\log Kn)$ time.

findOrd(π): Return a reverse valid ordering V_π of the nodes in π by scanning the nodes in $V_\mathcal{H}$ backwards (see Figure 4.4). Boolean $inPath$, for each node in $\mathcal{H}$, is used to indicate if a node is in π. The worst case complexity is $O(size(\mathcal{H}))$.

Procedure *K-BS* is shown in Figure 4.5. Line 2-4 finds the minimum weight hyperpath π^1, creates the root of the branching tree and inserts it into the heap. Line 5-18 contains the main loop. In the k'th iteration the minimal branching tree node τ is picked and removed from the heap and its corresponding hypergraph $\tilde{\mathcal{H}}$ built. Since we do not store the hyperpath corresponding to τ, we recalculate and output π^k on line 9 and next, a reverse valid ordering of π^k is found. The partitioning corresponding to Branching Operation 4.1.2 is performed on line 11-17. As pointed out, the path from branching tree node τ to the root of the branching tree defines end-tree $\tilde{\eta}$ and each s-t hyperpath in $\tilde{\mathcal{H}}$ must contain $\tilde{\eta}$. End-tree $\tilde{\eta}$ defines the set of nodes where the backward star have been fixed in previous branching operations and hence the subhypergraph, obtained by removing one of the hyperarcs in $\tilde{\eta}$, will contain no s-t hyperpath. Therefore these nodes are not

```
 1 procedure K-BS(H, s, t, K)
 2     SHTacyclic(s, H);
 3     if (W(t) < ∞) then insert(null, W(t));
 4     else STOP (there is no s-t hyperpath);
 5     for (k := 1 to K) do
 6         τ := delMin();
 7         if (τ = null) then STOP (there are no more s-t hyperpaths);
 8         H̃ := setF(τ);
 9         call SHTacyclic(s, H̃) and OUTPUT π^k;
10         (u_{q_k}, ..., u_1, s) := findOrd(π^k); p(u_{q_k+1}) := null;
11         for (i := q_k to 1) do
12             if (p(u_i) not fixed) then
13                 H̃ := modF(H̃, p(u_i), p(u_{i+1}));
14                 SHTacyclic(s, H_i^k);
15                 insert(p(u_i), W(t));
16             end if
17         end for
18     end for
19 end procedure
```

Figure 4.5: Finding the K best strategies.

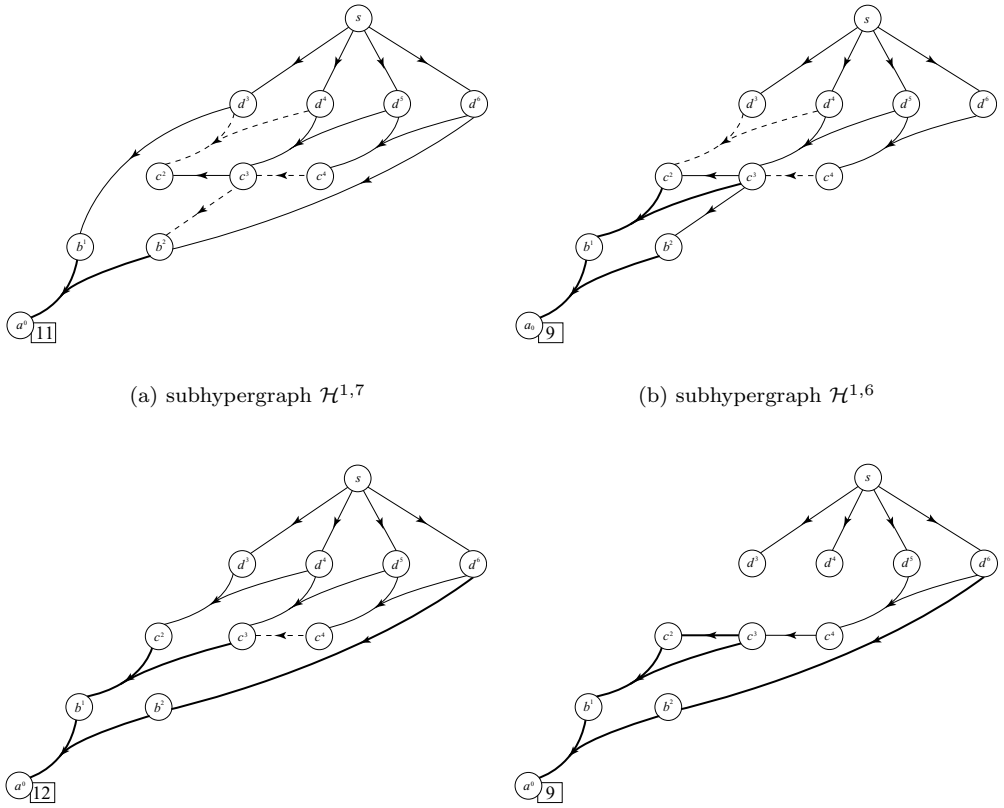
considered again (line 12). If $p^k(u_i)$ has not already been fixed, we create subhypergraph $\mathcal{H}^{k,i}$, find the minimum weight hyperpath and insert $\left(p^k(u_i), W(t)\right)$ into the branching tree and heap if $W(t) < \infty$ (line 13-15). Note, if $|BS_{\tilde{\mathcal{H}}}(p(u_i))| = 1$, then subhypergraph $\mathcal{H}^{k,i}$, obtained by removing $p(u_i)$, contains no s-t hyperpath. Therefore we do not have to use procedure *SHTacyclic* on line 14, we may just insert $(p(u_i), \infty)$ immediately on Line 15.

Since the k'th iteration partitions the set $\Pi^k \setminus \{\pi^k\}$ into subsets by considering the subhypergraphs $\mathcal{H}^{k,i}$ corresponding to Π_i^k, we have that the k'th iteration only removes π^k from Π^1. That is, at the end of the k'th iteration, the set of branching tree nodes in the heap represents the partition of $\Pi^1 \setminus \{\pi^1, ..., \pi^k\}$. Furthermore, at most n subhypergraphs are considered for each iteration. Therefore procedure *K-BS* must solve $O(Kn)$ minimum weight hyperpath problems and since the complexity of procedure *SHTacyclic* dominates the complexity of the other subprocedures, we have the following theorem.

Theorem 4.1.3 *Procedure K-BS finds the K minimum weight hyperpaths in worst time complexity $O(size(\mathcal{H}) \cdot Kn)$.*

Example 3 (continued) Consider the time-expanded hypergraph $\mathcal{H}$, shown in Figure 3.3 on page 28, where the minimal hypertree containing minimal s-a^0 hyperpath π^1 for the MMC problem is shown with solid lines. Assume that we want to find the $K = 3$ best strategies minimizing maximum cost, i.e. the distance weighting function is used. A valid ordering of π^1 where s precedes the other nodes can be found by ranking the nodes in $\mathcal{V} \setminus \{s\}$ in decreasing order of time, i.e. $V_\pi = (s, u_1, ..., u_8) = \left(s, d_,^6 d^5, d^4, c^3, c^2, b^2, b^1, a^0\right)$.

Hence using Branching Operation 4.1.2 on π^1 corresponds to creating subhypergraphs $\mathcal{H}^{1,8}, ..., \mathcal{H}^{1,1}$. subhypergraph $\mathcal{H}^{1,7}, ..., \mathcal{H}^{1,4}$ are shown in Figure 4.6 with the minimal



(a) subhypergraph $\mathcal{H}^{1,7}$

(b) subhypergraph $\mathcal{H}^{1,6}$

(c) subhypergraph $\mathcal{H}^{1,5}$

(d) subhypergraph $\mathcal{H}^{1,4}$

Figure 4.6: Subhypergraphs created when using Branching Operation 4.1.2.

hypertree shown with solid lines. The hyperarcs which are fixed are shown in bold. The remaining subhypergraphs are not shown, since, removing the hyperarc results in a subhypergraph where no s-a^0 hyperpath exists.

Assume that the minimum weight hyperpath in $\mathcal{H}^{1,4}$ is picked as the second minimum weight hyperpath π^2. Since nodes in $\mathcal{H}^{1,4}$ have only one hyperarc in its backward star, we have that none of the subhypergraphs obtained when using Branching Operation 4.1.2 on π^2 contain an s-a^0 hyperpath. Next the third minimum weight hyperpath is picked as the minimum weight hyperpath in $\mathcal{H}^{1,6}$. ■

4.1.2 Using reoptimization to reduce computation time

The main drawback of procedure *K-BS* is that a minimum weight hyperpath problem must be solved for each subhypergraph generated during the branching operation. The number of minimum weight hyperpath problems to solve is therefore much larger than K. In this section we use reoptimization techniques to avoid solving minimum weight

hyperpath problems, indeed we shall devise a specialized procedure where procedure *SHTacyclic* is called only once.

Let $W^k(v)$, $v \in \mathcal{V}$ denote the node weight of a minimum weight hypertree $\mathcal{T}_s^k$ in $\mathcal{H}^k$, containing the k'th minimum weight hyperpath π^k. Hyperpath π^k is defined by the predecessor function p^k of $\mathcal{T}_s^k$. Consider the subhypergraphs $\mathcal{H}^{k,i}$, $i = 1, ..., q_k$ corresponding to Branching Operation 4.1.2 on π^k.

Theorem 4.1.4 *Let $W^{k,i}(v)$, $v \in \mathcal{V}$, denote the weight of node v in a minimum weight hypertree in subhypergraph $\mathcal{H}^{k,i}$. Then $W^{k,i}(v) = W^1(v)$ for all nodes v below node u_i in the valid ordering $V_{\mathcal{H}}$. Moreover,*

$$W^{k,i}(u_i) = \min_{e \in BS_{\mathcal{H}^{k,i}}(u_i)} w(e) + F\left(W^1, e\right) \tag{4.1}$$

where $F(W^1, e)$ denotes the function $F(e)$ using weights $W^1(u)$, as defined in Section 2.3.1.

Proof Consider subhypergraph $\mathcal{H}^{k,i}$. Recall that no hyperarcs have been removed from the backward star of a node v below node u_i in the valid ordering $V_{\mathcal{H}}$ and therefore $W^{k,i}(v) = W^1(v)$. Furthermore, all nodes in $T(e)$, $e \in BS(u_i)$ are below u_i in $V_{\mathcal{H}}$. Hence, using procedure *SHTacyclic* in node u_i, corresponds to solving (4.1). ∎

Theorem 4.1.4 is useful, since, by storing the weights W^1 of the minimum weight hypertree of $\mathcal{H}$ we can avoid recalculating the weight of all nodes below node u_i in $V_{\mathcal{H}}$. According to the definition of $\mathcal{H}^{k,i}$ in Theorem 4.1.2, subhypergraph $\mathcal{H}^{k,i}$, $i = 1, ..., q_k$ satisfies

$$BS_{\mathcal{H}^{k,i}}(u_j) = p^k(u_j), \quad j = i+1, ..., q_k \tag{4.2}$$

That is, an *s-t* hyperpath in $\mathcal{H}^{k,i}$ must contain the end-tree $\eta^{k,i}$ defined in Definition 4.1.1 with inner-nodes I_η and leaf-nodes E_η. End-tree $\eta^{k,i}$ can be found by traversing the path from the branching tree node corresponding to $\mathcal{H}^{k,i}$ to the root of the branching tree. The predecessor function defining $\pi^{k,i}$ then satisfy

Theorem 4.1.5 *The predecessor of the minimum weight hyperpath $\pi^{k,i} = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ in $\mathcal{H}^{k,i}$ is equal to*

1. *Predecessor $p^k(v)$ for $v \in I_\eta$.*
2. *The predecessor defined by equation (4.1) for node u_i.*
3. *Predecessor $p^1(v)$ for $v \in \mathcal{V}_\pi \setminus (I_\eta \cup \{u_i\})$.*

Proof According to (4.2) the predecessor of $\pi^{k,i}$ is equal to $p^k(v)$ for $v \in I_\eta$ and the predecessor hyperarc of u_i is the hyperarc giving $W^{k,i}(u_i)$ in equation (4.1). Finally, since no hyperarcs have been removed from the backward star of a node v below node u_i, we have that the predecessor hyperarc is equal to $p^1(v)$. ∎

Given end-tree $\eta^{k,i}$, it is obvious that $u_i \in E_\eta$. Furthermore, u_i is the last node in a valid ordering of E_η. Therefore, using Corollary 2.4.1 for the value weighting function, and Corollary 2.4.2 for the distance weighting function we have

```
1  procedure setFP(τ)
2      for (u ∈ V \ {s}) do p(u) := p^1(u);
3      remove e_τ from H; v = h(e_τ);
4      while (e_τ ≠ null) do
5          τ := point(τ);
6          if (level changed) then remove e_τ from H;
7          else fix e_τ in H and set p(h(e_τ)) = e_τ;
8      end while
9      for (e ∈ BS(v)) do
10         W(v) := 0;
11         if (w(e) + F(W^1, e) < W(v)) then p(v) := e;
12     end for
13 end procedure
```

Figure 4.7: Subprocedure *setFP* of *K-BSreopt*.

Theorem 4.1.6 *The weight of the minimal hyperpath $\pi^{k,i}$ in $\mathcal{H}^{k,i}$ is equal to*

$$W^{k,i}(t) = W^{k}(t) + \left(W^{k,i}(u_i) - W^{k}(u_i)\right) f^{\eta}(u_i)$$

if the mean weighting function is considered, where f^{η} is defined as in (2.10) *for $\eta^{k,i}$. Similarly, the weight of the minimal hyperpath $\pi^{k,i}$ in $\mathcal{H}^{k,i}$ is equal to*

$$W^{k,i}(t) = \max\left\{W^{k}(t), W^{k,i}(u_i) + l^{\eta}(u_i)\right\}$$

if the distance weighting function is considered. Here l^{η} is defined as in (2.15) *for $\eta^{k,i}$.*

A K best strategies procedure using reoptimization can now be formulated. Each node in $\mathcal{H}$ contains labels $W^1(u)$ and $p^1(u)$. That is, the weights of a minimal hypertree in $\mathcal{H}$ and its corresponding predecessor function. Moreover, labels $p(u)$ and $f(u)$ are used at the current iteration of the procedure where $p(u)$ is used to store the current predecessor hyperarcs and $f(u)$ stores the value of f^{η} (or l^{η}). Two new subprocedures are used

setFP(τ): Shown in Figure 4.7. First, reset the labels $p(u)$ to $p^1(u)$. Next, build the subhypergraph corresponding to branching tree node $\tau = (e_\tau, W_\tau)$ by setting the flag in the hyperarcs as in procedure *setF*. At the same time if fix a hyperarc e_τ, then set $p(h(e_\tau)) = e_\tau$. Finally, find the new predecessor in the node where hyperarcs have been removed but not fixed. Given Theorem 4.1.5, we have that, when procedure *setFP* terminates, label p defines the hyperpath π^k in hypergraph $\mathcal{H}^k$. Since the procedure only removes a hyperarc once, worst case complexity becomes $O(m)$.

calc$W(u, e_r)$: First, find the weight $W^k(u_i)$ and then the weight defined in (4.1), i.e. scan the hyperarcs in $BS(u_i)$ of subhypergraph $\mathcal{H}^k$ to find $W^k(u_i)$. Then do the same with hyperarc e_r removed to find $W^{k,i}(u_i)$. Next, use Theorem 4.1.6 to find and return the weight in node t.

Procedure *K-BSreopt* for finding the K best strategies, i.e. the K minimum weight hyperpaths is shown in Figure 4.8. The procedure only differs from procedure *K-BS*

```
 1 procedure K-BSreopt(H, s, t, K)
 2     SHTacyclic(s, H);
 3     if (W(t) < ∞) then insert(null, W(t));
 4     else STOP (there is no s-t hyperpath);
 5     for (u ∈ V) do W^1(u) := W(u) and p^1(u) := p(u);
 6     for (k := 1 to K) do
 7         f(t) := 1; for (v ∈ V \ {t}) do f(v) := 0;
 8         τ := delMin();
 9         if (τ = null) then STOP (there are no more s-t hyperpaths);
10         setFP(τ);
11         OUTPUT π^k;
12         (u_{q_k}, ..., u_1, s) := findOrd(π^k); p(u_{q_k+1}) := null;
13         for (i := q_k to 1) do
14             for (v ∈ T(p(u_{i+1}))) do f(v) := f(v) + a_{p(u)}(v) f(u_{i+1});
15             if (p(u_i) not fixed) then
16                 insert(p(u_i), calcW(u_i, p(u_i)));
17             end if
18         end for
19     end for
20 end procedure
```

Figure 4.8: Finding the K best strategies using reoptimization.

in the way the predecessor function of π^k is found and how the weight in node t is calculated. In the k'th iteration we first initialize label f. By using procedure *setFP,* we have that labels $p(u)$ defines the predecessor function of hyperpath π^k in $\mathcal{H}^k$. On line 14, we update labels $f(u)$ so that they correspond to $f^{\eta}(u)$ of the current end-tree $\eta^{k,i}$, provided that the value weighting function is considered. For the distance function, we use $f(v) := \max(f(u_{i+1}) + w(p(u_{i+1})), f(v))$ instead. Procedure *calcW* finds the weight of the minimum weight hyperpath in $\mathcal{H}^{k,i}$.

Note that each time Branching Operation 4.1.2 is used in procedure *K-BSreopt*, we perform at most $O(size(\mathcal{H}))$ calculations using procedure *calcW*. Hence the worst case complexity of procedure *K-BSreopt* becomes linear in K.

Theorem 4.1.7 *Procedure K-BSreopt finds the K minimum weight hyperpaths in worst time complexity $O(size(\mathcal{H}) \cdot K)$.*

4.2 Finding the K best strategies under a priori route choice

Consider an STD network defined by the topological network G and the time-expanded hypergraph $\mathcal{H}$ with valid ordering

$$V_{\mathcal{H}} = (s = v_1, ..., v_n = t)$$

In this section we consider the problem of finding the K *best strategies* under *a priori* route choice between an *origin* node o and a *destination* node d in G when leaving the

origin at time zero. That is, we are interested in ranking the first K path-strategies in non-decreasing order using one of the criteria in Section 3.2. Note that, under a priori route choice, we seek path-strategies where the route must be specified before travel begins, i.e. we seek strategies corresponding to loopless *o-d* paths in G.

A priori route choice may for instance be useful for routing highly sensitive substances for which the travel path must be preapproved or where the driver does not have access to (or time to access) information while travelling.

Unfortunately, due to Theorem 3.3.2, finding the best path-strategy is $\mathcal{NP}$-hard. Therefore the problem of finding the K *best path-strategies* is also $\mathcal{NP}$-hard. However, experimental tests, provided in Section 4.3, show that the problem can be solved for relatively large networks.

Consider a hyperarc e in $\mathcal{H}$ not equal to a waiting or dummy arc. Then e corresponds to a unique arc a in G. Furthermore, a node $v \in \mathcal{V} \setminus \{s\}$ corresponds to a unique node in G. Let $arc\,(e)$, $e \in \mathcal{E}$ denote the arc in G corresponding to e. If e is a waiting or dummy arc in $\mathcal{H}$, we assume that $arc\,(e) = null$. Moreover, let $node\,(v)$, $v \in \mathcal{V} \setminus \{s\}$, denote the node in G corresponding to v in $\mathcal{H}$.

According to Corollary 3.3.2 there is a one to one correspondence between a path-strategy S_{o0} and an *s-t* hyperpath π with valid ordering $V_\pi \subseteq V_\mathcal{H}$ where t is the node in $\mathcal{H}$ corresponding to leaving the origin at time zero. Let the set of nodes in π corresponding to node u in G be

$$\mathcal{V}_\pi\,(u) = \{v \in \mathcal{V}_\pi : node\,(v) = u\}$$

Similar, $\mathcal{V}_\mathcal{H}\,(u)$ denote set of nodes in $\mathcal{H}$ corresponding to node u in G.

Consider two nodes in π corresponding to the same node in G. If the predecessor hyperarc of the nodes do not define a waiting arc, the predecessor must correspond to the same arc in G according to (3.4) resulting in the following corollary.

Corollary 4.2.1 *Hyperpath π defined by predecessor p corresponds to a loopless o-d path in G if for all $v, v' \in \mathcal{V}_\pi\,(u)$, we have*

$$arc\,(v) \neq null, arc\,(v') \neq null \Rightarrow arc\,(p\,(v)) = arc\,(p\,(v')) \tag{4.3}$$

Corollary 4.2.1 can be checked by starting in node t, proceeding down to node s and stop as soon as (4.3) fails. In this way, we build a subpath from the origin o to a node u in G. Even if a hyperpath does not satisfy Corollary 4.2.1 it is obvious that it still defines a subpath in G if (4.3) holds for a subset of the nodes in G.

Corollary 4.2.2 *Hyperpath π corresponds to travelling subpath $P_\pi = (u_1, a_1, ..., a_{q-1}, u_q)$ in G if for each node u_i, $1 \leq i \leq q$ we have*

$$arc\,(p\,(v)) = (u_i, u_{i+1})\,, \quad \forall v \in \mathcal{V}_\pi\,(u_i)\,, arc\,(p\,(v)) \neq null$$

Procedure *findSubP* shown in Figure 4.9 finds a loopless subpath P_π in G starting at the origin corresponding to π in $O\,(size\,(\mathcal{H}))$ time. The procedure starts in node o in G, i.e. node t in $\mathcal{H}$. In the **while** loop we consider the nodes in π corresponding to the last node u in P_π. If no successor arc is defined an arc $a \in A$ is found in line 6. Otherwise the remaining nodes in $\mathcal{V}_\pi\,(u)$ are examined and if all nodes have a predecessor corresponding to a or to a waiting arc, we add the arc to path P_π (line 12). Finally, if $h\,(a) = d$, we

```
1  procedure findSubP(π)
2      u := o; stop := false;
3      while (stop = false) do
4          a := null;
5          for (v ∈ V_π(u)) do
6              if (a = null and arc(p(v)) ≠ null) then a := arc(p(v));
7              else if (arc(p(v)) ≠ null and arc(p(v)) ≠ a) then
8                  stop := true and break;
9              end if
10         end for
11         if (stop = false) then
12             P_π := P_π ∪ (u, a, h(a)); u := h(a);
13             if (u = d) then stop := true;
14         end if
15     end while
16 end procedure
```

Figure 4.9: A procedure finding a subpath defined by π.

have that π corresponds to a loopless o-d path, i.e. π defines a path-strategy and the procedure stops; otherwise the loop is repeated on $h(a)$.

In the next section we consider the problem of finding the K best path-strategies in an STD network where *no waiting* is allowed in the nodes in G. Different procedures are presented distinguished by the way a lower bound on the weight of a best path-strategy is found. In Section 4.2.2 we extend the results to STD networks where *waiting* is allowed.

4.2.1 No waiting allowed

Consider an STD network where *no waiting* is permitted in the nodes in G and let $\mathcal{P}$ denote the set of *loopless* o-d paths in G. Then a traveller following a path $P \in \mathcal{P}$ must upon arrival at an intermediate node u in P leave u immediately. Hence path P defines a unique arc $a \in A$ followed for each node $u \in P$ and leaving time.

Corollary 4.2.3 *A path $P \in \mathcal{P}$ corresponds to a unique path-strategy S_{o0}, i.e. a unique s-t hyperpath π in $\mathcal{H}$ and conversely a path-strategy corresponds to a unique path in G.*

Due to Corollary 4.2.3, we have that finding the K best path-strategies corresponds to finding the K minimum weight paths in G where the minimum weight of a path P is the weight of the hyperpath corresponding to P using one of the criteria in Section 3.2. Therefore we partition $\mathcal{P}$ into subsets, i.e. the branching operation is not performed on the hyperpath π but on a subpath in G corresponding to π. More specifically, let π denote a minimum weight hyperpath in $\mathcal{H}$. First, check if π defines a path-strategy using procedure *findSubP*. Two cases may arise:

1. Procedure *findSubP* returns a loopless o-d path[‡] $P_\pi = (o = u_1, ..., u_q = d)$, i.e. the best path-strategy has been found corresponding to P_π.

[‡]The arcs in the path are not written explicitly.

2. Procedure *findSubP* considers a node v in $\mathcal{H}$ where (4.3) fails, i.e. π defines a subpath $(o = u_1, ..., u_{q-1} = v)$. In this case, let $P_\pi = (o = u_1, ..., u_{q-1}, u_q)$ denote the subpath together with arc (u_{q-1}, u_q) where (u_{q-1}, u_q) denotes the arc a in procedure *findSubP* when (4.3) fails (line 7).

The idea is now to partition $\mathcal{P}$ using P_π. That is, branch on the path from u_1 to u_q found above, starting at u_1 and proceeding forward. This is the usual forward branching approach of Yen applied to P_π in G with a special treatment of node u_q.

Branching Operation 4.2.1 *Given path $P_\pi = (o = u_1, ..., u_q)$ in G, the set $\mathcal{P}$ can be partitioned into disjoint subsets $\mathcal{P}^i$, $1 \le i \le q$ as follows:*

1. *For $1 \le i < q$, paths in $\mathcal{P}^i$ contain path $P^i_\pi = (u_1, ..., u_i)$ but not arc (u_i, u_{i+1}).*
2. *Paths in $\mathcal{P}^q$ contain path P_π.*

Proof It is obvious that the partition of $\mathcal{P}$ defines a branching tree similar to the one shown in Figure 4.1 and hence $\mathcal{P}^i$, $1 \le i \le q$ are disjoint and $\mathcal{P} = \cup_{i=1,...,q}\mathcal{P}^i$. ■

Note, if $u_q = d$, then $\mathcal{P}^q$ contains a single path, namely P_π, i.e. in this case P_π may be removed from $\mathcal{P}$ by not considering $\mathcal{P}^q$.

Definition 4.2.1 Given P_π, let subgraph G^i, $i = 1, ..., q$ be obtained from G as follows

1. For each node u_j, $j = 1, ..., i-1$, remove each arc in $FS(u_j)$ except (u_j, u_{j+1}).
2. If $i \neq q$, remove arc (u_i, u_{i+1}).

Definition 4.2.2 Given P_π, let subhypergraph $\mathcal{H}^i$, $i = 1, ..., q$ be obtained from $\mathcal{H}$ as follows

1. For each node $v \in \mathcal{V}_\mathcal{H}(u_j)$, $j = 1, ..., i-1$, remove each hyperarc in $BS(v)$ except the hyperarc e with $arc(e) = (u_j, u_{j+1})$.
2. If $i \neq q$, then for each node $v \in \mathcal{V}_\mathcal{H}(u_i)$, remove the hyperarc $e \in BS(v)$ with $arc(e) = (u_i, u_{i+1})$.

Note that, Branching Operation 4.2.1, Definition 4.2.1 and Definition 4.2.2 only consider the partition of G into subgraphs. However, it is obvious that we can use Branching Operation 4.2.1 recursively.

The partition of $\mathcal{P}$ corresponds to a partition of G into subgraphs G^i or equivalent to partition $\mathcal{H}$ into subhypergraphs $\mathcal{H}^i$.

Theorem 4.2.1 *Given subgraph G^i and subhypergraph $\mathcal{H}^i$ for $i = 1, ..., q$, the following are equivalent*

1. *$P_{od} \in \mathcal{P}^i$.*
2. *P_{od} is an o-d path in G^i.*
3. *s-t hyperpath π corresponding to path P_{od} is a hyperpath in $\mathcal{H}^i$.*

Proof Consider $P_{od} \in \mathcal{P}^i$. Note that all arcs in P_{od} is in G^i, i.e.P_{od} is a path in G^i. Next, assume that P_{od} is a path in G^i. Note that $\mathcal{H}^i$ contains all hyperarcs corresponding to an arc in P_{od}. Therefore the s-t hyperpath π corresponding to P_{od} is contained in $\mathcal{H}^i$. Finally, consider an s-t hyperpath $\pi \subseteq \mathcal{H}^i$ corresponding to P_{od}. Since the predecessor hyperarc e of a node v in π with $node\,(v) = u_j$, $j = 1, ..., i-1$ is a hyperarc with $arc\,(e) = (u_j, u_{j+1})$, we have that π defines subpath $P^i_\pi = (u_1, ..., u_i)$. Moreover, if $i < q$ the predecessor hyperarc e of a node v satisfying $node\,(v) = u_i$ cannot correspond to arc (u_i, u_{i+1}), i.e. P_{od} cannot contain arc (u_i, u_{i+1}), i.e. $P_{od} \in \mathcal{P}^i$. ■

Using Theorem 4.2.1, each subset $\mathcal{P}^i$ can be represented by its corresponding subgraph G^i. Furthermore, since the weight of the best strategy is a lower bound on the weight of the best path-strategy we have that a lower bound on the minimum weight path in $\mathcal{P}^i$ can be found by considering subhypergraph $\mathcal{H}^i$.

Corollary 4.2.4 *The weight of the minimum weight hyperpath in subhypergraph $\mathcal{H}^i$ corresponding to subgraph G^i is a lower bound on the weight of the minimal path in $\mathcal{P}^i$.*

Recall that, in Branching Operation 4.2.1, branching is performed on a subpath in G, not on a hyperpath in $\mathcal{H}$. Since the number of arcs in path P_π found, when examining the minimum weight hyperpath π, is expected to be much smaller than the number of hyperarcs in π, the number of subsets $\mathcal{P}^i$ is expected to be smaller than if branching was performed on π instead. Furthermore, branching is not necessarily performed on an o-d path. If only a subpath P_π is found when examining the minimum weight hyperpath π (i.e. $u_q \neq d$), the last arc (u_{q-1}, u_q) in P_π is found where (4.3) fails. It is then obvious that hyperpath π is not contained in $\mathcal{H}^i$, $i = 1, ..., q$. That is, we can stop the branching operation in node u_q, because the partition removes the current minimum weight hyperpath π. On the contrary, if $u_q = d$, we have found a loopless o-d path in G and its corresponding path-strategy defined by π.

Procedure *K-BPS*, shown in Figure 4.10, for finding the K best path-strategies can now be formulated. A backward and forward representation of the time-expanded hypergraph $\mathcal{H}$ and the topological network G is used. Furthermore, $\mathcal{H}$ and G are linked together with pointers, so that, for each hyperarc e and node u in $\mathcal{H}$, $arc\,(e)$ and $node\,(u)$ can be identified. For further details see Appendix B.1.4. A candidate set of pairs $(\tilde{lb}, \tilde{G})$ representing the current partition of $\mathcal{P}$ is used. Graph $\tilde{G}$ is the subgraph representing subset $\tilde{\mathcal{P}}$ and $\tilde{lb}$ is the weight of the minimal hyperpath in subhypergraph $\tilde{\mathcal{H}}$ corresponding to $\tilde{G}$. That is, $\tilde{lb}$ is a lower bound on the weight of the minimal path in $\tilde{G}$. Note that it is still possible to maintain the partition of $\mathcal{P}$ implicitly in a branching tree similar to the one shown in Figure 4.2. The branching tree node corresponding to $(\tilde{lb}, \tilde{G})$ contains pair $(\tilde{a}, \tilde{lb})$, where $\tilde{a}$ is the hyperarc we remove in the forward star. Moreover, if the pair corresponds to subset $\mathcal{P}^q$ in Branching Operation 4.2.1, then we fix the whole path P_π. In this case we set $\tilde{a} = null$. A heap of pointers to the branching tree nodes is used to maintain the candidate set in non-decreasing order of the lower bound. Procedure *K-BPS* uses the following subprocedures:

SHTacyclic$(s, \mathcal{H}, \tilde{G})$: Finds the minimum weight hyperpath in subhypergraph $\tilde{\mathcal{H}}$ corresponding to $\tilde{G}$. Done by modifying line 4 in procedure *SHTacyclic*$(s, \mathcal{H})$, shown in Figure 2.4 on page 14, to only considering hyperarcs e in $BS\,(v_i)$ satisfying that $arc\,(e)$ is in $\tilde{G}$ or e is a dummy arc.

```
1  procedure K-BPS(H, G, s, t, K)
2      SHTacyclic(s, H);
3      if (W(t) = ∞) then STOP (there is no path-strategy);
4      insert(null, W(t)); k := 0
5      while (k < K) do
6          τ := delMin();
7          if (τ = null) then STOP (there are no more path-strategies);
8          G̃ := setF(τ);
9          call SHTacyclic(s, H, G̃) to find π̃;
10         P_π̃ = (o = u_1, a_1, ..., a_{q-1}, u_q) := findP(π̃, a_q); a_0 := null;
11         if (u_q = d) then
12             k := k + 1, OUTPUT the k'th path-strategy π̃ and set j := q − 1;
13         else P_π̃ := P_π̃ ∪ a_q, j := q + 1 and a_{q+1} := null;
14         for (i := 1 to j) do
15             if (a_i not fixed) then
16                 G̃ := modG(G̃, a_i, a_{i−1});
17                 SHTacyclic(s, H, G̃);
18                 insert(a_i, W(t));
19             end if
20         end for
21     end while
22 end procedure
```

Figure 4.10: Finding the K best path-strategies.

findP$(\tilde{\pi}, \tilde{a})$: Return the path found in procedure *findSubP* in Figure 4.9. Moreover, the arc $\tilde{a}$ used as input in *findP* is set to *null* if path P_π is an o-d path; otherwise if the procedure stops before reaching node d, arc $\tilde{a}$ is set to the arc a used in procedure *findSubP*, i.e. the arc used when (4.3) fails (line 7).

setF(τ): Similar to procedure *setF* under time-adaptive route choice. Subgraph $\tilde{G}$ can be obtained from G by maintaining a flag for each arc in G equal to one, if the arc is in subgraph $\tilde{G}$ and zero otherwise. An arc (u, v) in G is then fixed by setting the flags of the hyperarcs in $FS(u)$ to zero except the flag for arc (u, v). Moreover, note that, o-d paths in G must contain the fixed subpath $(o, ..., u, v)$. Since we seek loopless o-d paths, we also remove the arcs $a \in BS(v) \setminus \{(u, v)\}$.

modG$(\tilde{G}, a_r, a_f)$: Modify subgraph $\tilde{G}$ by removing arc a_r and next fix arc a_f (if not equal to *null*).

At the start of an iteration the minimal branching tree node τ is picked and removed from the heap. Subgraph $\tilde{G}$ is then built by traversing the path from τ to the root in the branching tree. The minimum weight hyperpath in subhypergraph $\tilde{\mathcal{H}}$, corresponding to $\tilde{G}$, is found using procedure *SHTacyclic*$(s, \mathcal{H}, \tilde{G})$. Given $\tilde{G}$, we find the minimum weight hyperpath $\tilde{\pi}$ of $\tilde{\mathcal{H}}$ and use procedure *findP* to find path $P_{\tilde{\pi}} = (o = u_1, ..., u_q)$. If $u_q = d$, then we have found a best path-strategy and output it. In this case the subgraph defined by fixing all arcs in $P_{\tilde{\pi}}$ is equal to $P_{\tilde{\pi}}$ and hence we only insert the first $q - 1$ subgraphs

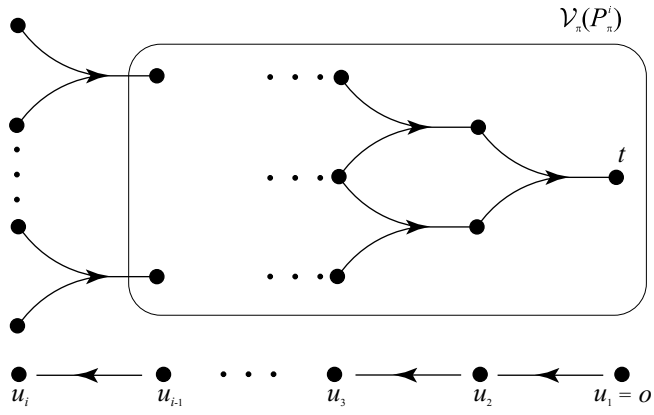


Figure 4.11: The end-tree η^i defined by subpath P^i_π.

into the branching tree and heap (line 12). If $u_q \neq d$ we have to branch on the arc where (4.3) fails. Therefore path $P_{\tilde{\pi}}$ is modified on line 13 and we consider the partition of $\tilde{G}$ into $q+1$ subgraphs. For each subgraph $\tilde{G}$ we find the lower bound $\tilde{lb}$ on the weight of the minimal path in $\tilde{G}$ and insert the pair into the branching tree and candidate set.

Note that the subgraphs $\tilde{G}$ defined in the branching tree are ranked according to a lower bound, which are equal to the weight of the minimal hyperpath in the subhypergraph corresponding to $\tilde{G}$. Therefore the weight of the minimal branching tree node in the heap is equal to the weight of the minimal hyperpath $\tilde{\pi}$ defined by all the subgraphs in the candidate set. That is, if $\tilde{\pi}$ defines a path-strategy, i.e. an o-d path in G, we have found the minimal one. Moreover, due to Branching Operation 4.2.1, we only remove an o-d path from G if $\tilde{\pi}$ defines a path-strategy (line 12). That is, the following theorem holds

Theorem 4.2.2 *Procedure K-BPS finds the K best path-strategies in a finite number of steps.*

The worst case complexity of the procedure is exponential, since the problem of finding the best path-strategy is $\mathcal{NP}$-hard.

Ranking nodes in the branching tree using a weaker lower bound

Consider subgraph $\tilde{G}$ corresponding to a branching tree node in the branching tree. In procedure *K-BPS* each branching tree node is ranked according to a lower bound on the weight of the minimal path in $\tilde{G}$, equal to the weight of the minimal hyperpath in the subhypergraph $\tilde{\mathcal{H}}$ corresponding to $\tilde{G}$. That is, a minimum weight hyperpath problem must be solved for each subgraph generated during the branching operation. However, we may use reoptimization techniques to find another lower bound. In general, this lower bound is weaker, since it is a lower bound on the weight of the minimal hyperpath in $\tilde{\mathcal{H}}$. However, we do not need to solve a minimum weight hyperpath problem for each subgraph generated during the branching operation.

Consider Branching Operation 4.2.1 on path $P_\pi = (o = u_1, ..., u_q)$ found by considering s-t hyperpath π. In each subgraph G^i, $i = 1, ..., q$ we have fixed the u_1-u_i subpath P^i_π

of P_π. According to Definition 4.2.2, fixing the subpath from u_1 to u_i corresponds to fixing the predecessor from t and backwards in $\mathcal{H}^i$ (see Figure 4.11). That is, a hyperpath in $\mathcal{H}^i$ must contain the hypergraph as shown in Figure 4.11. More formally, let $\mathcal{V}_\pi(P^i_\pi)$ denote the set of nodes in π corresponding to nodes in P^i_π except node u_i. Consider subhypergraph $\eta^i = (\mathcal{E}_\eta, \mathcal{V}_\eta)$ of $\mathcal{H}^i$, $i = 2, ..., q-1$ with

$$\mathcal{E}_\eta = \bigcup_{u \in \mathcal{V}_\pi(P^i_\pi)} \{p(u)\}, \quad \mathcal{V}_\eta = \bigcup_{e \in \mathcal{E}_\eta} (T(e) \cup h(e))$$

Since there is a path from each node $u \in \mathcal{V}_\eta$ to node t in η^i, we have, according to Definition 2.4.1, that

Corollary 4.2.5 *Subpath P^i_π corresponds to the unique end-tree $\eta^i \subset \pi$ defined by $\mathcal{V}_\pi(P^i_\pi)$, $i = 2, ..., q-1$.*

Note that we do not define η^q because arc (u_{q-1}, u_q) may be an arc where (4.3) is violated. That is, η^q is an end-tree but may not be contained in π. End-tree η^i has a particular structure, indeed the set of inner-nodes is equal to $\mathcal{V}_\pi(P^i_\pi)$ and the set of leaf-nodes E_η is contained in $\mathcal{V}_\pi(u_i)$. However, since we branch on G and not on $\mathcal{H}$, nodes in $\mathcal{H}$ are not processed backward according to the valid ordering of π. That is, we cannot use the weights of a minimal hypertree $\mathcal{T}_s$ to find the weight of the minimal hyperpath to each node in E_η. But by using the weights of the minimal hypertree $\mathcal{T}_s$ containing the s-t hyperpath π used in Branching Operation 4.2.1, we can find a lower bound on the minimum weight hyperpath in subhypergraph $\mathcal{H}^i$.

Due to Theorem 4.1.4, it is obvious that a lower bound $lb^i(u)$ for each node $u \in E_\eta$ in $\mathcal{H}^i$ is

$$lb^i(u) = \min_{e \in BS_{\mathcal{H}^i}(u)} w(e) + F\left(W^{\mathcal{T}_s}, e\right) \tag{4.4}$$

where $F(W^{\mathcal{T}_s}, e)$ denotes the function $F(e)$ using the weights of $\mathcal{T}_s$, as defined in Section 2.3.1. Furthermore, given the lower bounds $lb^i(u)$, $u \in E_\eta$, the lower bound for node t can be calculated using Theorem 4.1.6 on end-tree η^i. For subhypergraph $\mathcal{H}^1$ no end-tree is defined. However, here we only remove the hyperarc $p(t)$, i.e. the lower bound in node t is found using (4.4). Furthermore, no end-tree is defined for $\mathcal{H}^q$, but end-tree η^{q-1} is also an end-tree in $\mathcal{H}^q$, i.e. we just use (4.4) on η^{q-1} in $\mathcal{H}^q$.

A procedure *K-BPSreopt*, finding the K best path-strategies using reoptimization, can now be formulated. The procedure only differs slightly from procedure *K-BPS*, therefore the pseudo-code of procedure *K-BPSreopt* is not shown. Since branching tree nodes are ranked according to a lower bound, which is not equal to the weight of the minimal hyperpath, two cases may happen when we pick and remove a branching tree node τ from the heap:

1. The weight W_τ of the minimal hyperpath π_τ in the subhypergraph corresponding to τ is lower than or equal to the current minimal lb in the candidate set.

2. The weight W_τ of the minimal hyperpath in the subhypergraph corresponding to τ is greater than the current minimal lb in the candidate set.

In the first case, π_τ is the hyperpath with minimum weight among all the subhypergraphs in the candidate set and we proceed as in procedure *K-BPS*. In the second case,

this is not necessarily true. Therefore we reinsert τ into the candidate set with the lower bound updated to W_τ. That is, we do not perform any branching operation but start on a new iteration. Furthermore, a label for each node in $\mathcal{H}$ must be maintained containing the value of l^η or f^η for the end-tree as in procedure *K-BSreopt*. Finally, on line 17 in procedure *K-BPS*, we find a lower bound in node t as descried above, instead of using procedure *SHTacyclic*.

Using a multiple branching approach

In this section a more restricted branching operation is presented. Consider Branching Operation 4.2.1, *o-d* paths in subset $\mathcal{P}^i$ must contain subpath $P^i_\pi = (u_1, ..., u_i)$ and not arc (u_i, u_{i+1}). Note, however, we do not take into consideration that a path in $\mathcal{P}^i$ must have a unique successor arc of node u_i. This can be done by considering each arc a in the forward star of u_i separately. That is, instead of creating one subset $\mathcal{P}^i$ where we remove the successor arc (u_i, u_{i+1}), we create $|FS(u_i)| - 1$ subsets, where we fix an arc $a \in FS(u_i)$ satisfying $a \neq (u_i, u_{i+1})$.

More specifically, let π denote a minimum weight hyperpath in $\mathcal{H}$ and assume that π defines subpath $P_\pi = (o = u_1, ..., u_q)$. Note that P_π differs from the path used in Branching Operation 4.2.1. Here we only consider the subpath defined by π. Now, the idea is to partition $\mathcal{P}$ using the following *multiple branching operation* on P_π.

Branching Operation 4.2.2 *Given path $P_\pi = (o = u_1, ..., u_q)$ defined by π the set $\mathcal{P}$ can be partitioned into disjoint subsets as follows:*

1. *For $1 \leq i < q$ and $a \in FS(u_i) \setminus \{(u_i, u_{i+1})\}$, paths in $\mathcal{P}^{i,a}$ contain path P^i_π and arc a.*

2. *If $u_q \neq d$, then for $a \in FS(u_q)$, paths in $\mathcal{P}^{q,a}$ contain path P_π and arc a, otherwise $\mathcal{P}^{q-1,(u_{q-1},u_q)} = \{P_\pi\}$.*

Proof A branching tree similar to the one shown in Figure 4.1 can be constructed where each level, instead of having a remove and a fix arc, has a fix arc for each arc in the forward star. Note, if $u_q = d$, we do not consider the subset $\mathcal{P}^{q-1,(u_{q-1},u_q)}$ in item *1* above, therefore it is defined in item *2*. ■

Note that $\mathcal{P}^{i,a}$ may be empty, for instance this is the case if the head node of arc a is in path P^i_π. Similar to Definition 4.2.1 and Definition 4.2.2, the subset $\mathcal{P}^{i,a}$ corresponds to subgraph $G^{i,a}$ and a subhypergraph $\mathcal{H}^{i,a}$.

Definition 4.2.3 Given P_π and subset $\mathcal{P}^{i,a}$, let subgraph $G^{i,a}$ be obtained from G as follows

1. For each node u_j, $j = 1, ..., i-1$, remove each arc in $FS(u_j)$ except (u_j, u_{j+1}).

2. Remove each arc in $FS(u_i)$ except arc a.

Definition 4.2.4 Given P_π and subset $\mathcal{P}^{i,a}$, let subhypergraph $\mathcal{H}^{i,a}$ be obtained from $\mathcal{H}$ as follows

1. For each node $v \in \mathcal{V}_{\mathcal{H}}(u_j)$, $j = 1, ..., i-1$, remove each arc in $BS(v)$ except the hyperarc e with $arc(e) = (u_j, u_{j+1})$.

2. For each node $v \in \mathcal{V}_{\mathcal{H}}(u_i)$, remove each arc in $BS(v)$ except the hyperarc e with $arc(e) = a$.

Theorem 4.2.3 *Given subset $\mathcal{P}^{i,a}$, subgraph $G^{i,a}$ and subhypergraph $\mathcal{H}^{i,a}$ the following are equivalent*

1. *$P_{od} \in \mathcal{P}^{i,a}$.*

2. *P_{od} is an o-d path in $G^{i,a}$.*

3. *Hyperpath π in $\mathcal{H}^{i,a}$ defines a path-strategy corresponding to P_{od}.*

Proof Similar to the proof of Theorem 4.2.1. ∎

A procedure finding the K best path-strategies using the multiple branching operation, denoted *K-BPS_MB*, can now be formulated. The procedure only differ from the procedure *K-BPSreopt* in the way the subgraphs $\tilde{G}$ are made. Moreover, a slightly different branching tree implementation must be used (see Appendix B.2). Note that a hyperpath in subhypergraph $\mathcal{H}^{i,a}$ must contain end-tree η^i and hence a lower bound on the weight of the best path-strategy for each subhypergraph $\mathcal{H}^{i,a}$ can be found by using (4.4) and Theorem 4.1.6 on end-tree η^i.

Using Branching Operation 4.2.2 creates more branching tree nodes compared to Branching Operation 4.2.1. However, we obtain a tighter lower bound and often the size of the forward star of a node in G is small in practice (about four in a road network). Therefore, if the calculation of the lower bound for each subgraph is fast, we may obtain a faster procedure.

4.2.2 Waiting allowed

Consider a strategy S_{o0} defining the route followed between node o and d when leaving node o at time zero. Assume that S_{o0} is a path-strategy, i.e. S_{o0} defines a path $P \in \mathcal{P}$.

$$P = (o = u_1, a_1, u_2, a_2, ..., a_{q-1}, u_q = d)$$

If no waiting is allowed, then path-strategy S_{o0} corresponding to P is unique according to Corollary 4.2.3. However, if waiting is allowed, then a traveller arriving at node u_i, $1 < i < q$, following path P, can either wait or leave along arc a_i. That is, if waiting is allowed, then P defines a possible exponential number of path-strategies, distinguished from each other by the use of waiting. Therefore there may exist an exponential number of hyperpaths corresponding to P each having a different weight.

Due to this, the problem of finding the K best strategies under a priori route choice when waiting is allowed can be divided into two cases depending on whether the K best path-strategies must correspond to K different *o-d* paths in G, or if the path-strategies not necessarily have to correspond to different *o-d* paths in G. Note that in this case we may find K best path-strategies corresponding to the same path in G.

We start by looking at the first problem and next proceed to show that the latter problem can be solved by a simple extension of the first problem.

```
1  procedure calcLB(H̃, P̃ = (u_1, ..., u_i))
2      for (v ∈ V_H̃(u_j), j = 1, ..., i) do W(v) := ∞;
3      for (j := i to 1) do
4          V_{N_H̃(u_j)} := (v_1, ..., v_{r(u_j)});
5          for (l := 1 to r(u_j)) do
6              for (e ∈ BS(v_l)) do
7                  W(v_l) := min(W(v_l), w(e) + F(e));
8              end for
9          end for
10     end for
11 end procedure
```

Figure 4.12: Finding a lower bound (waiting allowed).

Paths must be different

It is easy to see that Branching Operation 4.2.1 still holds and the construction of the subgraphs G^i defined in Definition 4.2.1 is not affected by waiting. However, the hypergraph $\mathcal{H}^i$ defined in Definition 4.2.2 does not contain all hyperpaths corresponding to a path in $\mathcal{P}^i$, since we remove the waiting arc when we fix a hyperarc. Instead another set of subhypergraphs has to be constructed.

Definition 4.2.5 Given path P_π in Branching Operation 4.2.1, let subhypergraph $\mathcal{H}^i$, $i = 1, ..., q$ be obtained from $\mathcal{H}$ as follows

1. For each node $v \in \mathcal{V}_{\mathcal{H}}(u_j)$, $j = 1, ..., i-1$, remove each hyperarc in $BS(v)$ except the hyperarc e with $arc(e) = (u_j, u_{j+1})$ and the unique waiting arc (if it exists).

2. For each node $v \in \mathcal{V}_{\mathcal{H}}(u_i)$, remove the hyperarc e with $arc(e) = (u_i, u_{i+1})$.

Given $\mathcal{H}^i$ in Definition 4.2.5 it is easy to see that Theorem 4.2.1 still holds. Moreover, if P_π defines an *o-d* path, then all *s-t* hyperpaths in $\mathcal{H}^q$ correspond to P_π. Hence, by removing the branching tree node corresponding to $\mathcal{H}^q$ from the candidate set, we remove all path-strategies corresponding to P_π. Procedure *K-BPS* in Figure 4.10 can now be easily modified to find the K minimum weight paths under waiting. We only have to modify Subprocedure $SHTacyclic(s, \mathcal{H}, \tilde{G})$ so that we only consider hyperarcs which are either equal to a waiting arc, dummy arc or correspond to an arc in $\tilde{G}$. Hence we find the minimum weight of a hyperpath in a subhypergraph as defined in Definition 4.2.5 and if P_π is an *o-d* path then subhypergraph $\mathcal{H}^q$ is removed from the candidate set, i.e. we only output one path-strategy for each path in G, namely the best one.

Unfortunately, fixing the subpath P^i_π in G^i does not define an end-tree in $\mathcal{H}^i$ in the waiting case. Therefore, procedure *K-BPSreopt* cannot be used directly. However, equation (4.4) still can be used to find a lower bound on all the nodes $v \in \mathcal{V}_{\mathcal{H}}(u_i)$.

Procedure *calcLB*, shown in Figure 4.12, can be used to find a lower bound on the minimal hyperpath in subhypergraph $\tilde{\mathcal{H}}$ where each hyperpath must contain the subpath $\tilde{P} = (u_1, ..., u_i)$. Procedure *calcLB* is a modified version of procedure *SHTacyclic* which only considers the nodes in $\tilde{\mathcal{H}}$ corresponding to nodes in $\tilde{P}$. First, the node weights are

initialized to infinity. We next process the nodes according to a valid ordering $V_{\tilde{\mathcal{H}}}$ of $\tilde{\mathcal{H}}$. Every hyperpath in $\tilde{\mathcal{H}}$ must define subpath $\tilde{P}$, hence, by considering node u_i down to the first node u_1 in P_π^i, we follow a valid ordering of $\tilde{\mathcal{H}}$. Note that, given node u_j, we must follow a valid sub-ordering $V_{\mathcal{V}_{\mathcal{H}}(u_j)} \subset V_{\tilde{\mathcal{H}}}$ of the nodes in $\mathcal{V}_{\mathcal{H}}(u_j)$ defined on line 4. This ordering is maintained in the data structures representing G and $\mathcal{H}$ (see Appendix B.1.4). Label $W(t)$ then contains a lower bound on the weight of the minimal hyperpath in $\tilde{\mathcal{H}}$ when the procedure stops.

Similarly the multiple Branching Operation 4.2.2 still holds under waiting and we have to redefine the subhypergraphs $\mathcal{H}^{i,a}$.

Definition 4.2.6 Given P_π and subset $\mathcal{P}^{i,a}$, let subhypergraph $\mathcal{H}^{i,a}$ be obtained from $\mathcal{H}$ as follows

1. For each node $v \in \mathcal{V}_{\mathcal{H}}(u_j)$, $j = 1, ..., i-1$, remove each arc in $BS(v)$ except the hyperarc e with $arc(e) = (u_j, u_{j+1})$ and the unique waiting arc (if it exists).

2. For each node $v \in \mathcal{V}_{\mathcal{H}}(u_i)$, remove each arc in $BS(v)$ except the hyperarc e with $arc(e) = a$ and the unique waiting arc (if it exists).

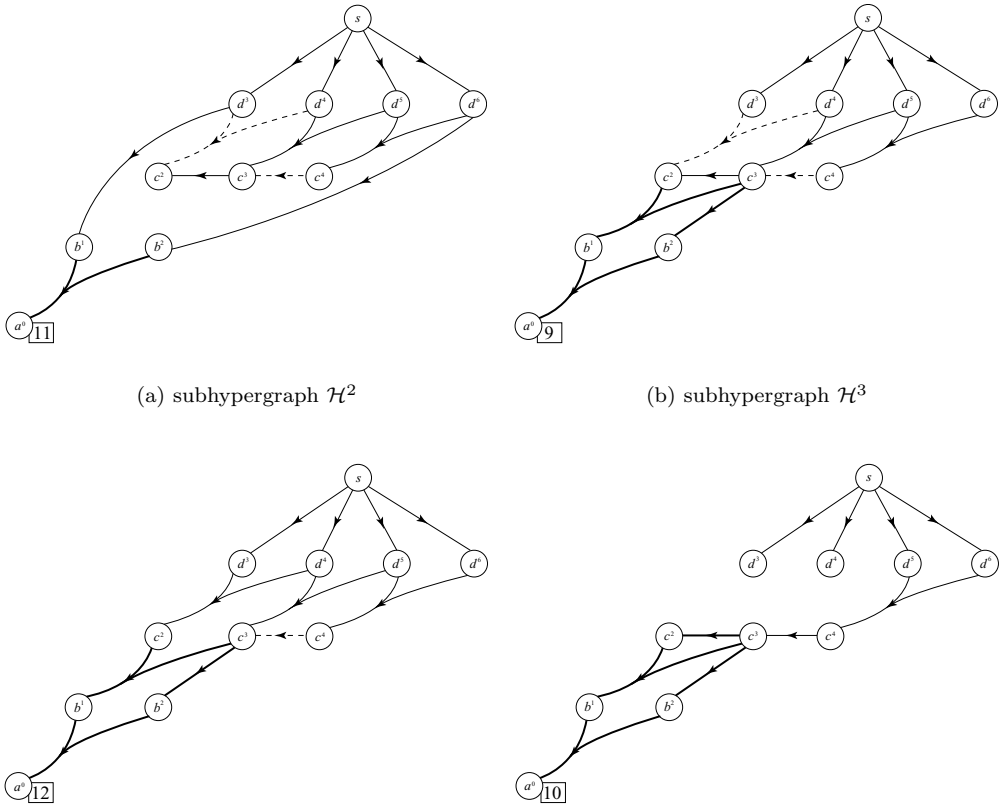
Given $\mathcal{H}^{i,a}$ in Definition 4.2.6, we have that Theorem 4.2.3 still holds; therefore procedure *K-BPS_MB*, in the waiting case, can use subprocedure *calcLB* described above on $\mathcal{H}^{i,a}$ defined in Definition 4.2.6.

Paths does not have to be different

We consider the problem of finding the K best path-strategies not necessarily corresponding to K different paths in G. Consider the path P_π used in Branching Operation 4.2.1 or Branching Operation 4.2.2 obtained from the minimum weight hyperpath π in $\mathcal{H}$. If π does not define a path-strategy then the partition of $\mathcal{P}$ does not remove any paths from the candidate set and the procedures run like in the previous section. If π defines a path-strategy, then P_π is an *o-d* path. In this case P_π is removed from the candidate set in the procedures used when no waiting is allowed. However, if waiting is allowed, there may be exponentially many different path-strategies corresponding to P_π and we are not interested in removing P_π from $\mathcal{P}$ but only in removing the path-strategy, i.e. hyperpath π. This can be done in the following way.

First, consider the procedures using Branching Operation 4.2.1, if P_π is an *o-d* path, then all *s-t* hyperpaths in subhypergraph $\mathcal{H}^q$ correspond to P_π, i.e. we are interested in removing π from $\mathcal{H}^q$. However, since all *s-t* hyperpaths in $\mathcal{H}^q$ correspond to P_π, we can remove π from $\mathcal{H}^q$ by using Branching Operation 4.1.2. That is, we perform a backward branching operation using the hyperarcs in π not the arcs in P_π. In this way we partition $\mathcal{H}^q$ into subhypergraphs $\mathcal{H}^{q,j}$ and for each subhypergraph, we now can find the minimum weight hyperpath which defines the best path-strategy in $\mathcal{H}^{q,j}$. Using a slightly modified branching tree (see Appendix B.2), we can now store the minimum weight and $\mathcal{H}^{q,j}$ in the candidate set. Note if $\mathcal{H}^{q,j}$ later is picked from the candidate set, then all *s-t* hyperpaths in $\mathcal{H}^{q,j}$ define an *o-d* path and we may use Branching Operation 4.1.2 directly on $\mathcal{H}^{q,j}$.

Finally, consider the procedures using Branching Operation 4.2.2. If P_π is an *o-d* path, then all *s-t* hyperpaths in subhypergraph $\mathcal{H}^{q-1,(u_{q-1},u_q)}$ correspond to P_π and Branching Operation 4.1.2 is used on $\mathcal{H}^{q-1,(u_{q-1},u_q)}$ as described above.



(a) subhypergraph $\mathcal{H}^2$

(b) subhypergraph $\mathcal{H}^3$

(c) subhypergraph $\mathcal{H}^{3,4}$

(d) subhypergraph $\mathcal{H}^{3,3}$

Figure 4.13: Subhypergraphs created when using Branching Operation 4.2.1 and Branching Operation 4.1.2.

Example 3 (continued) Consider the topological network G in Figure 3.2 on page 27. Note that there are only two a-d paths in G. However, since waiting is possible in node c, they define five path-strategies – one corresponding to path (a, b, d) and four corresponding to path (a, b, c, d).

Assume that we want to find the $K = 3$ best path-strategies minimizing maximum cost using procedure *K-BPS*.

First the minimal hypertree for the MMC problem is found in the time-expanded hypergraph $\mathcal{H}$, containing the minimal s-a_0 hyperpath π, shown with solid lines in Figure 3.3 on page 28.

Next, procedure *findP* finds the path $P_\pi = (a, b)$. Since P_π is not an a-d path, we use Branching Operation 4.2.1 on $P_\pi := (a, b, c)$. Note P_π is the union of the subpath (a, b) defined by π and a successor arc of node b where equation (4.3) fails. Another successor arc may be chosen, e.g. arc (b, d). Subgraphs G^i, $i = 1, 2, 3$, is created and the lower bound found. The subhypergraphs $\mathcal{H}^2$ and $\mathcal{H}^3$ corresponding to G^2 and G^3 are shown

in Figure 4.13(a) and 4.13(b) where hyperarcs in the minimal hypertree is shown with solid lines and the hyperarcs corresponding to a fixed arc in G are shown in bold. The subhypergraph of G^1 is not shown, since no s-a_0 hyperpath exists.

The minimum weight hyperpath π^1 in $\mathcal{H}^3$ is then picked from the candidate set. Since π^1 corresponds to an a-d path $P_{\pi^1} = (a, b, c, d)$, we output π^1 as the first best path-strategy. Consider the subgraphs $G^{3,i}$, $i = 1, 2, 3, 4$ of G^3. In all subgraphs $G^{3,i}$, $i = 1, 2, 3$, no a-d path exist, i.e. nothing is added to the candidate set. However, waiting is allowed in node c, and hence we have to use Branching Operation 4.1.2 on π^1 in the subhypergraph corresponding to $G^{3,4}$ which, in this case, is equal to $\mathcal{H}^3$ (see Figure 4.13(b)). A valid ordering of π^1, where s precedes the other nodes, can be found by ranking the nodes in $\mathcal{V}\backslash\{s\}$ in decreasing order of time:

$$V_{\pi^1} = (s, u_1, ..., u_7) = \left(s, d^5, d^4, c^3, c^2, b^2, b^1, a^0\right)$$

Hence using Branching Operation 4.1.2 on π^1 corresponds to creating subhypergraphs $\mathcal{H}^{3,i}$, $i = 1, ..., 7$ of $\mathcal{H}^3$. Only subhypergraphs $\mathcal{H}^{3,4}$ and $\mathcal{H}^{3,3}$ contain an s-a_0 hyperpath. The subhypergraphs are shown in Figure 4.13(c) and 4.13(d) where the hyperarcs fixed in V_{π^1} are shown in bold. The second best path-strategy is then found as the minimum weight hyperpath π^2 in subhypergraph $\mathcal{H}^{3,3}$. All nodes in $\mathcal{H}^{3,3}$ have only one hyperarc in their backward star and hence there will be no s-a_0 hyperpath in the subhypergraphs of $\mathcal{H}^{3,3}$. The third best path-strategy is then the minimum weight hyperpath in $\mathcal{H}^2$. Note the first two best path-strategies correspond to path (a, b, c, d) while the third corresponds to path (a, b, d). ■

4.3 Computational results

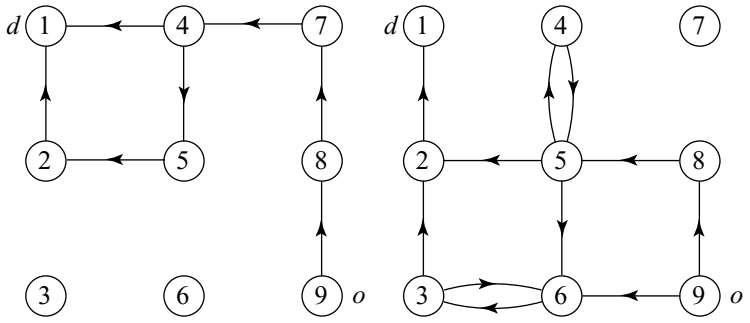
In this section we report the computational experience with the procedures previously described in this chapter. The procedures have been implemented in C++ and tested on a 1 GHz PIII computer with 1GB RAM using a Linux Red Hat operating system. The programs have been compiled with the GNU C++ compiler with optimize option -O.

All tests are performed on time-expanded hypergraphs generated with the TEGP generator (see Section 3.5). In the following, we use the term *hypergraph class* to define a particular setting of the TEGP input parameters; for each class, different hypergraphs (i.e. different instances of the problem) can be generated by choosing different seeds.

For each hypergraph class, five independent runs was carried out using a different seed. In all classes, a cycle consists of 144 time instances, i.e. 12 hours divided in 5 minute intervals. A cycle has two peaks each with a total length of 5 hours (each period pk_j of the peak lasts 1 hour and 40 minutes) and the first peak starts after half an hour ($t = 6$). The interval of possible off-peak mean travel times is $[lb_t, ub_t] = [2, 6]$, i.e. an off-peak mean travel time between 10 and 30 minutes. The deviation mean ratio is set to $\rho = 0.25$ in all classes. The remaining input parameters will be specified when we consider the different classes used in the tests.

4.3.1 Performance measures/statistics

In this section, performance measures/statistics used to evaluate the procedures, are described. The statistics can be divided into two groups. In group one statistics concerning



(a) "path-like". (b) Not very "path-like".

Figure 4.14: A "path-like" and a not very "path-like" strategy.

the performance of the procedures are considered. For each hypergraph class the measures are averaged over five independent runs using different seeds. The abbreviation used in the tables are given in parentheses.

CPU time (CPU_K): CPU time for finding the K best strategies (time-adaptive case) or path-strategies (a priori case). The running time does not include input/output time.

First CPU time (CPU_1): CPU time for finding the best strategy or path-strategy.

Number of iterations (ite_K): Number of iterations performed for finding the K best path-strategies, i.e. the number of times the **while** loop in e.g. procedure *K-BPS* is executed. Only used under a priori route choice since the number of iterations under time-adaptive route choice is equal to the number of strategies K found.

Number of iterations (ite_1): Number of iterations performed for finding the best path-strategy.

Number of reinsertions (*reins*): The number of times a branching tree node picked from the candidate set is reinserted into the candidate set. Only used under a priori route choice, since no reinsertions are made under time-adaptive route choice. Reported in percent of the number of iterations ite_K.

Average number of branching tree nodes ($|\tau|$): The average number of branching tree nodes τ created when performing a branching operation. That is, under time-adaptive route choice, the number of hyperarcs in the hyperpath not already fixed, and under a priori route choice, the number of arcs in the path P_π used in the branching operation not already fixed.

In the second group, statistics on how the strategies look like, are given, e.g. number of hyperarcs in the hyperpaths, increase in the weight between the first and the K'th strategy etc. Here we also examine how "path-like" the strategies are – defined in the

following sense: Given a hyperpath π defining a strategy S, it is obvious that π defines a subgraph $G_\pi = (N_\pi, A_\pi)$ of G with

$$N_\pi = \{u = node(v) \mid v \in \mathcal{V}_\pi\}, \quad A_\pi = \{a = arc(e) \mid e \in \mathcal{E}_\pi\}$$

Each node u in N_π is a node which may be visited when following strategy S and each arc a in $FS(u)$ is an arc followed for some leaving time t from node u. Moreover, if the number of arcs in $FS(u)$ is equal to one, the same arc is followed for all leaving times from u. If this holds for all $u \in N_\pi$, then S defines a path in G. On the contrary, if the number of arcs in $FS(u)$ is higher than one in most for the nodes in N_π, then strategy S defines a strategy which is not very "path-like", since different routes are followed depending on the actual travel time on the arcs. A path-like and not very path-like strategy is shown in Figure 4.14 when considering a 3×3 grid network G. Note that strategies may define cycles in G_π corresponding to travelling from a node at time t and arriving at the same node at time $t' > t$.

We use the following statistics to see how strategies look like. Again, for each hypergraph class, the measures are average over five independent runs using a different seed and the abbreviation used in the tables is given in parentheses.

Relative increase in weight (*inc*): The relative increase between the first strategy and the K'th strategy under time-adaptive route choice and under a priori route choice between the first path-strategy and the K'th path-strategy. Reported in percent.

Relative increase strategy to path-strategy (inc_{S-PS}): The relative increase between the first strategy and the first path-strategy reported in percent. Only reported under a priori route choice.

Average number of hyperarcs in π ($|\mathcal{E}_\pi|$): The average number of hyperarcs in a specific set $\tilde{\Pi}$ of hyperpaths. Under time-adaptive route choice, $\tilde{\Pi}$ is equal to the hyperpaths used by the branching operation. Under a priori route choice, $\tilde{\Pi}$ is equal to the hyperpaths used by the branching operation not corresponding to a path-strategy.

Average number of arcs ($|A_\pi|$): The average number of arcs in the subgraphs G_π corresponding to the hyperpaths in the set $\tilde{\Pi}$ described above.

Average percentage of nodes with j *successor*(s) $(s = j)$: For each subgraph G_π corresponding to a hyperpath in $\tilde{\Pi}$, the percentage of nodes with j successors is found. Note that, since the TEGP generator considers grid networks, a node in the subgraph has at most four successors. The average over the subgraphs G_π is reported for $j = 1, 2, 3$ and 4.

Number of different paths (*diff*): Used under a priori route choice when waiting is allowed. Report the number of different paths the K path-strategies correspond to.

4.3.2 Time-adaptive route choice

Our main goal here is to evaluate and compare the behavior of procedure *K-BS* and *K-BSreopt*. Both the MEC and the MMC criterion is considered, i.e. we are minimizing

Class	1	2	3	4	5	6	7	8	9	10	11
n	1586	1975	2320	2682	4157	1497	1497	1497	1497	1497	1497
m_h	5309	6608	7738	8955	13907	5012	5012	5012	5012	5012	5012
m_a	47	60	71	77	111	44	44	44	44	44	44
$\|T(e)\|$	3	3	4	5	5	3	3	3	3	3	4
H	78	94	118	133	155	75	75	75	75	75	75
I_T	[2,8]	[2,11]	[2,15]	[2,19]	[2,23]	[2,8]	[2,8]	[2,8]	[2,8]	[2,8]	[2,8]
I_C	[1,1100]	[1,1500]	[1,2000]	[1,2500]	[1,3000]	[1,1050]	[1,1100]	[1,1200]	[1,1500]	[1,2000]	[1,2000]
ψ	0.1	0.5	1.0	1.5	2.0	0	0	0	0	0	-
r_ξ	0	0	0	0	0	0.05	0.10	0.20	0.50	1.00	-

Table 4.1: Class 1-11 used for preliminary testing (grid size 5×10).

expected or maximum possible cost. Time criteria will not be considered, since similar results are obtained.

First, some preliminary tests are performed to examine how changes in the structure of the costs of the hyperarcs may effect how path-like the strategies are. An underlying grid size of 5×10 is used and eleven hypergraph classes are considered. In Table 4.1, the following statistics for class 1-11 are reported:

Number of nodes (n): Recall that the generator may generate nodes and hyperarcs in $\mathcal{H}$ which cannot be contained in an s-t hyperpath. These nodes and hyperarcs are removed from $\mathcal{H}$ in a preprocessing step. The number of nodes in $\mathcal{H}$ after preprocessing are reported.

Number of hyperarcs (m_h): Number of "true" hyperarcs in $\mathcal{H}$ after preprocessing (i.e. $|T(e)| > 1$).

Number of arcs (m_a): Number of arcs in $\mathcal{H}$ after preprocessing.

Tail size ($|T(e)|$): The average number of nodes in the tail of each hyperarc, i.e. the average number of elements in the travel time density of $X(u, v, t)$ when leaving node u at time t along arc (u, v).

Time horizon (H): The time horizon or number of time instances of the STD network.

Travel time interval (I_T): The interval of possible travel times. Note that this interval may not be equal to $[lb_t, ub_t]$, since the travel time increases in peaks, if peak dependent travel times/costs are used (see Section 3.5).

Cost interval (I_C): The interval of costs used. Note that this interval may not be equal to $[lb_c, ub_c]$, since the costs increase in peaks, if peak dependent costs are used (see Section 3.5).

The value of the peak increase parameter ψ and the range of the random perturbation r_ξ is also reported in Table 4.1.

Class	CPU_K	$\|\tau\|$	$\|\mathcal{E}_\pi\|$	$\|A_\pi\|$	$s=1$	$s=2$	$s=3$	$s=4$	Class	CPU_K	$\|\tau\|$	$\|\mathcal{E}_\pi\|$	$\|A_\pi\|$	$s=1$	$s=2$	$s=3$	$s=4$
				MEC									MMC				
1	62	27	226	23	75	18	7	0	1	284	72	202	17	84	15	0	0
2	109	32	272	24	73	16	9	2	2	437	89	237	19	83	14	3	0
3	166	40	356	27	66	22	10	1	3	767	110	297	20	77	20	3	0
4	201	39	389	27	66	22	11	1	4	617	73	317	22	76	21	3	0
5	518	51	552	32	57	32	10	1	5	1193	115	464	24	71	25	4	0
6	84	20	227	22	74	20	4	1	6	282	74	196	16	86	13	1	0
7	132	32	245	24	71	22	7	0	7	261	68	196	18	82	17	1	0
8	115	29	265	27	65	26	9	0	8	281	73	222	20	76	21	3	0
9	237	58	312	35	53	33	13	1	9	271	71	265	29	60	31	9	0
10	319	82	401	54	38	39	18	4	10	286	77	371	48	50	32	16	2
11	1001	264	854	123	10	32	40	18	11	479	114	545	76	24	58	14	3

Table 4.2: Results for finding the $K = 1000$ best strategies using procedure *K-BS* (no waiting).

In class 1-10, peak dependent costs are used with an off-peak cost interval $[lb_c, ub_c] = [1, 1000]$. Changes in the peak increase parameter ψ and the range of the random perturbation r_ξ is tested separately.

For class 1-5, no random perturbation is used and the peak increase parameter ψ increases from 10% to 200%. Note that, increasing the value of ψ, increase the interval of possible travel times, the interval of costs used and the average tail size. Moreover, the time horizon grows, since, for larger travel times, we need a larger time horizon, i.e. the hypergraphs grow in size.

In class 6-10, the peak increase parameter ψ is set to zero and the range of the random perturbation r_ξ increases from 5% to 100%. Since the peak increase parameter is zero, no changes in time horizon are needed, i.e. the topological structure for the hypergraphs in class 6-10 is the same. However, the cost interval grows.

Finally, in class 11, random costs are generated in the interval $[1, 2000]$, i.e. the topological structure for the hypergraphs in class 11 are the same as in class 6-10. However, in class 6-10 the costs are peak dependent while the costs are random in class 11 (see Section 3.5).

The results for finding the $K = 1000$ best strategies using procedure *K-BS* on hypergraph class 1-11 are reported in Table 4.2 for the MEC and MMC case. First, consider the MEC criterion, i.e. the mean weighting function is used. Increasing the peak increase parameter, makes the strategies more non path-like: The average number of hyperarcs in the 1000 hyperpaths/strategies grows. Similarly does the average number of arcs in the subgraph G_π corresponding to hyperpath π. Moreover, the percentage of nodes in G_π with two, three or four successor arcs grows while the percentage of nodes in G_π with one successor arc falls. This is a result of the fact that, increasing ψ, makes the costs of a node in G for two different leaving times vary more. One may believe that the strategies become more non path-like as a result of the hypergraphs becoming larger in size when increasing ψ. However, this is not the case, as can be seen in the results of class 6-10.

	peak dependent costs			random costs		
Class	12	13	14	15	16	17
Grid size	5×10	10×10	20×10	5×10	10×10	20×10
n	2320	7573	21454	1497	3961	11856
m_h	7738	27177	79423	5012	14236	43903
m_a	71	101	147	44	59	88
$\|T(e)\|$	4	4	4	3	3	3
H	118	156	237	75	101	155
I_T	[2,15]	[2,15]	[2,15]	[2,8]	[2,8]	[2,8]
I_C	[2,2190]	[1,2197]	[1,2200]	[1,2000]	[1,2000]	[1,2000]

Table 4.3: Class 12-17 used to compare procedure *K-BS* and *K-BSreopt*.

Here the topological properties are the same for class 6-10, but increasing the range of the random perturbation, makes the strategies even more non path-like compared to class 1-5.

Finally, making the costs random have a big impact on the strategies. Most of the nodes in G_π now have three successors (40%) and the number of arcs in G_π increases significantly. Making the strategies more non path-like, affects the performance of procedure *K-BS*. The CPU time varies between 62-1001 seconds depending on how path-like the strategies are, even if the same grid size is considered in all classes. This is due to the fact that strategies not very path-like correspond to hyperpaths containing more hyperarcs, i.e. the branching operation has to be performed on more subhypergraphs and since procedure *K-BS* has to solve a minimum weight hyperpath problem for each subhypergraph, the CPU time may increase significantly.

For the MMC criterion, the same results are valid. However, the increase in ψ and r_ξ do not affect the strategies as much as under the MEC criterion. Moreover, note that when peak dependent costs are considered, the number of branching tree nodes created $|\tau|$ is larger than under the MEC criterion even though the number of hyperarcs in the hyperpaths is slightly smaller. This indicate that we branch on hyperpaths where the number of hyperarcs already fixed is smaller than under the MEC criterion.

The preliminary tests indicate that the structure of the costs may have a big impact on procedure performance. Therefore we test procedure *K-BS* and *K-BSreopt* on two groups of hypergraphs. In the first group, we use peak dependent costs and a combination of the peak increase parameter ψ and the random perturbation to model "realistic" STD networks and to show that our algorithms are expected to work well in practice. In group two we use random costs. The STD networks generated when using random costs do probably not correspond to "realistic" STD networks. However, these networks are used to show how the procedures work on difficult problem instances.

Six classes are used to test procedures *K-BS* and *K-BSreopt*. Statistics for class 12-17 are reported in Table 4.3. Class 12-14 use peak dependent costs. The range of the random perturbation is set to $r_\xi = 10\%$ and the peak increase parameter to $\psi = 100\%$. This gives a cost structure as shown in Figure 3.5 on page 30. Furthermore, different grid sizes are used, namely 5×10, 10×10 and 20×10. Therefore the time horizon increases with grid size.

Class 15-17 use random costs, i.e. there are no peak effect and no random perturbation,

Class	CPU_K (reopt)	CPU_K (K-*BS*)	CPU_1	$\lvert\tau\rvert$	$\lvert\mathcal{E}_\pi\rvert$	$\lvert A_\pi\rvert$	$s=1$	$s=2$	$s=3$	$s=4$	*inc*
MEC											
12	9.46	164.50	0.01	33	340	26	65	24	9	2	0.00
13	24.79	1521.61	0.03	88	895	54	60	29	10	1	0.00
14	70.17	6564.36	0.07	135	1688	76	65	31	4	0	0.00
15	7.26	812.17	0.01	264	854	123	10	32	40	18	0.00
16	25.87	4371.57	0.02	442	2162	253	6	24	41	29	0.00
17	75.79	45013.99	0.05	1401	7063	614	3	14	36	46	0.00
MMC											
12	9.14	569.16	0.01	99	320	21	74	23	3	0	0.00
13	24.82	4174.09	0.03	203	651	35	72	21	6	1	0.00
14	75.00	24706.39	0.08	400	1603	71	71	26	3	0	0.00
15	7.80	400.37	0.01	114	545	76	24	58	14	3	0.00
16	22.08	2773.09	0.02	251	1398	160	17	55	24	4	0.00
17	71.42	46471.10	0.06	1339	5215	405	10	40	39	11	0.00

Table 4.4: Results for finding the $K = 1000$ best strategies using procedure *K-BS* and *K-BSreopt* (no waiting).

instead costs are generated randomly in $[1, 2000]$. Grid sizes as in class 12-14 are used.

The results for finding the $K = 1000$ best strategies are reported in Table 4.4 for the MEC and MMC criteria. The CPU time for finding the 1000 best strategies is reported for both procedure *K-BSreopt* and *K-BS*. The rest of the statistics are the same for both procedures. It is clear that procedure *K-BSreopt* outperforms procedure *K-BS*. This is due to the face that procedure *K-BS* has to solve a minimum weight hyperpath problem for each subhypergraph generated during the branching operation, i.e. procedure *K-BS* must solve $|\tau|$ minimum weight hyperpath problems on average each time the branching operation is used. Since the average number of hyperarcs in the hyperpaths increase with grid size, resulting in an increase in $|\tau|$, this has a large effect on procedure *K-BS*, in particular when random costs are used. In procedure *K-BSreopt*, we for each branching operation on π, on the contrary, have to find a valid ordering of the nodes in π which takes $O\left(size\left(\mathcal{H}\right)\right)$, update the weights in the end-tree which takes $O\left(size\left(\pi\right)\right)$ time and finally, for each subhypergraph $\tilde{\mathcal{H}}$, we must scan $BS\left(u_i\right)$ to find the new weight in node u_i used to find the new weight in node t. That is, the calculation of new node weights takes on average $O(|BS_{\tilde{\mathcal{H}}}\left(u_i\right)| \cdot \text{average tail size} \cdot |\tau|)$ time. Therefore both the size of $\mathcal{H}$, π and $|\tau|$ will have an impact on the CPU time for procedure *K-BSreopt*. This can be seen on the results of class 12-17, if we consider the peak dependent and random costs separately. However, the CPU time for peak dependent and random costs are approximately the same when using procedure *K-BSreopt*. Note that the size of the hypergraph, when using random costs, is approximately half the size of the hypergraph when using peak dependent costs. This fact will make the CPU time fall under random costs. On the other hand, a larger size of the hyperpaths (column $|\mathcal{E}_\pi|$) and the branching nodes $|\tau|$ will make the CPU time rise under random costs. These two conflicting effects, seem to neutralize each other and as a result, the CPU time is approximately the same for procedure *K-BSreopt*

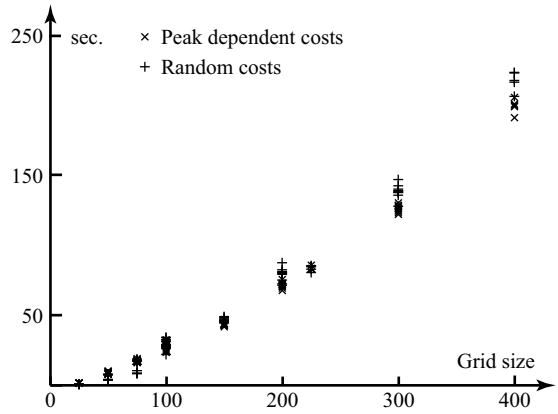


Figure 4.15: CPU times for increasing grid sizes using the MEC criterion ($K = 1000$).

no matter if we are considering peak dependent or random costs.

The CPU time for finding the first strategy is negligible, i.e. a minimum weight hyperpath problem can be solved very fast.

If we consider the average number of hyperarcs $|\mathcal{E}_\pi|$ in the hyperpaths, we see that $|\mathcal{E}_\pi|$ is large compared to $|\tau|$. Therefore a lot of hyperarcs are already fixed when performing a branching operation.

Finally, note that the increase in the weight between the first strategy and the 1000'th strategy is very small (below 0.00%). This behavior can be intuitively explained for both the MEC and MMC criterion.

First, consider the MEC criterion and a hyperpath π. The weight of π can be found using Theorem 2.3.1 and f^π. By looking at the recursive equations (2.4) defining f^π it is easy to see that $f^\pi(v)$ may be very small for some node v in π. This is true, in particular, if the hyperarc size is large and π contains long s-t paths. Let $W(v)$ denote the weight in node v of hyperpath π. Moreover, let $\tilde{W}(v)$ denote the weight in node v of the hyperpath $\tilde{\pi}$ obtained by changing the predecessor of node v. According to Corollary 2.4.1 the weight of hyperpath $\tilde{\pi}$ is then

$$\tilde{W}(t) = W(t) + \left(\tilde{W}(v) - W(v)\right) f^\pi(v)$$

In this situation, for a sufficiently small $f^\pi(v)$ any choice of predecessor in node v would give almost the same weight of π and $\tilde{\pi}$. Thus we can expect a huge number of hyperpaths with more or less the same weight.

For the MMC criterion there may exist a lot of hyperpaths with the same weight. Consider hyperpath π and a node v in π with predecessor hyperarc e. Assume that $u \in T(e)$ is a node used to find $W(v)$, that is, $W(v) = W(u) + w(e)$. Now suppose that hyperpath $\tilde{\pi}$ is obtained by changing the predecessor in a node $u' \in T(e)$, $u' \neq u$. The change in the predecessor of u' does not affect the weight in node v, unless $\tilde{W}(u')$ becomes greater than $W(u)$. In other words, we may have a lot of hyperpaths with exactly the same weight.

The fact that the increase in the weight is very small, will have a negative effect on the procedures used when considering bicriterion route choice in STD networks, as we will

	peak dependent costs			random costs		
Class	18	19	20	21	22	23
n	2329	2329	2329	1499	1499	1499
m_h	7756	7756	7756	5016	5016	5016
m_a	2328	2328	2328	1498	1498	1498
$\|T(e)\|$	4	4	4	3	3	3
H	118	118	118	75	75	75
I_T	[2,15]	[2,15]	[2,15]	[2,8]	[2,8]	[2,8]
I_C	[1,2190]	[1,2190]	[1,2190]	[1,2000]	[1,2000]	[1,2000]
I_W	[1,500]	[500,1000]	[1000,2000]	[1,500]	[500,1000]	[1000,2000]

Table 4.5: Class 18-23 where waiting is allowed (grid size 5×10).

see in the next chapter. Here we use a K best strategies procedure to find all strategies below a certain upper bound. However, since the increase in the weight is small, we have to find a lot of strategies before the upper bound is reached.

In order to investigate the behavior of the CPU time of procedure *K-BSreopt* more carefully, procedure *K-BSreopt* was tested for all grid sizes between 5×5 and 20×20 (increasing the grid base or height with 5 each time). The results for the MEC criterion is shown in Figure 4.15 on the page before using both peak dependent and random costs. Each point (grid size, CPU) in the figure corresponds to the CPU time of procedure *K-BSreopt* on a hypergraph in a class using one seed, i.e. five points are plotted for each class. The grid size denotes the number of nodes in the grid. The CPU time for random and peak dependent costs seems to be the same. Moreover, the CPU time seems to grow in a slightly curved line with the grid size. This is expected since doubling the grid size makes the corresponding time-expanded hypergraph triple in size (recall that the CPU time is affected by $size\,(\mathcal{H})$).

Waiting

Procedure *K-BSreopt* was also tested on STD networks where waiting is allowed. Six classes were used, namely classes 18-23. In all the classes a grid size of 5×10 is used. Statistics for the classes are reported in Table 4.5. Class 18-20 use peak dependent costs as in class 12 and three waiting costs intervals, namely low waiting costs $[1, 500]$, middle waiting costs $[500, 1000]$ and high waiting costs $[1000, 2000]$. In class 21-23 random costs are used as in class 15 with the same waiting cost intervals as in class 18-20.

The results for finding the $K = 1000$ best strategies are reported in Table 4.6 for the MEC and MMC criteria. If we compare the results with class 12 and class 15, we see that the CPU time is slightly smaller due to the fact that $|\tau|$ is smaller than if waiting is not allowed. Moreover, note that low waiting gives more nodes with one successor compared to when no waiting is allowed.

4.3.3 A priori route choice

We consider the problem of finding the K best strategies under a priori route choice, i.e. the K best path-strategies. Our main goal here is to evaluate and compare the behavior of

Class	CPU_K	CPU_1	$\|\tau\|$	$\|E_\pi\|$	$\|A_\pi\|$	$s=1$	$s=2$	$s=3$	$s=4$	inc
MEC										
18	7.90	0.00	32	351	23	75	15	9	1	0.00
19	7.88	0.00	21	338	25	68	18	12	2	0.00
20	7.88	0.00	19	339	26	68	18	12	2	0.00
21	3.70	0.00	59	476	76	25	54	18	4	0.00
22	7.38	0.01	134	776	117	10	35	40	14	0.00
23	7.46	0.00	164	794	117	13	31	39	18	0.00
MMC										
18	7.83	0.00	47	278	19	79	18	4	0	0.00
19	7.92	0.00	94	323	22	73	23	4	0	0.00
20	7.92	0.00	89	328	23	71	25	4	0	0.00
21	7.15	0.01	80	319	59	46	45	9	1	0.00
22	6.69	0.01	86	518	77	25	53	18	4	0.00
23	6.71	0.01	79	543	73	27	57	15	2	0.00

Table 4.6: Results for finding the $K = 1000$ best strategies using procedure *K-BSreopt* (waiting allowed).

procedure *K-BPS*, *K-BPSreopt* and *K-BPS_MB*. Both the MEC and the MMC criterion are considered, i.e. we are minimizing expected or maximum cost.

First, consider the preliminary tests on hypergraph class 1-11 under time-adaptive route choice. The results pointed out that different cost structures will affect how path-like the strategies are. Increasing the peak increase parameter and the range of the random perturbation when peak dependent costs are considered, will make the strategies more non path-like. Furthermore, using random costs have a big impact on the strategies which become even more non path-like.

Obviously the extent of how path-like the strategies are, will affect the procedures under a priori route choice, since, if strategies are not path-like, then the length of the path P_π used in the branching operation will be small. Hence when using random costs we may have to perform a lot of branching operations before a path-strategy is found. On the other hand if the strategies are more path-like then the length of P_π will be long and we will find a path-strategy faster.

Due to the comments above we will use hypergraph classes 12-23 to test procedures *K-BPS*, *K-BPSreopt* and *K-BPS_MB* where classes with peak dependent costs are used to test "realistic" STD networks and to show that our algorithms are expected to work in practice. The classes using random costs are used to show how the procedures will work on very difficult problem instances. It is expected that, finding the K best strategies under a priori route choice, will be much harder than the corresponding problem under time-adaptive route choice. Therefore only $K = 100$ path-strategies are found in the tests.

The results for finding $K = 100$ path-strategies in class 12-17 using procedure *K-BPS* are reported in Table 4.7. First, note that, as expected, the strategies in class 12-14 are more path-like (peak dependent costs) compared to class 15-17 (random costs).

Second, let us compare the results with the results under time-adaptive route choice.

Class	ite_K	CPU_K	ite_1	CPU_1	$\lvert\tau\rvert$	$\lvert\mathcal{E}_\pi\rvert$	$\lvert A_\pi\rvert$	$s=1$	$s=2$	$s=3$	$s=4$	inc	$inc_{S\text{-}PS}$
MEC													
12	356	8	13	0.3	3	233	22	82	14	4	1	49	5
13	454	37	11	1.0	4	411	31	82	14	4	0	17	7
14	2359	875	133	53.5	4	892	46	85	13	2	0	10	5
15	1427	25	133	2.5	2	203	46	50	25	20	5	18	43
16	9942	564	2722	157.9	2	475	90	39	21	27	12	8	54
17	203479	34227	89519	15127.5	2	1748	244	25	16	27	31	3	59
MMC													
12	258	9	4	0.2	3	246	19	86	12	2	0	48	4
13	405	52	14	2.2	4	422	27	85	13	2	0	17	5
14	3253	1330	250	108.0	4	816	44	86	12	2	0	10	9
15	839	15	109	2.0	2	231	32	64	28	7	1	12	25
16	4972	286	1487	87.0	2	509	56	54	32	13	1	5	33
17	579906	77120	293709	45262.1	2	1485	122	44	29	22	5	2	39

Table 4.7: Results for finding the $K = 100$ best path-strategies using procedure *K-BPS* (no waiting).

Under a priori route choice, the increase in weight for the first 100 path-strategies is between 2% and 49%, while, under time-adaptive route choice, the increase in the weight for the first 1000 strategies was below 0.00%. This will have a positive effect on the procedures used when considering bicriterion route choice in STD networks, as we will see in the next chapter. Here we use a K best path-strategies procedure to find all path-strategies below a certain upper bound. Furthermore, the branching tree size under a priori route choice is much smaller than under time-adaptive route choice. Recall that we have to create a subhypergraph for each hyperarc not fixed in the hyperpath when we perform the branching operation under time-adaptive route choice, while we only have to create a subgraph for each arc in the path P_π when we perform the branching operation under a priori route choice.

Next, let us compare the results when using peak dependent costs and random costs. Random costs have a negative impact on procedure performance. The number of iterations and the CPU time increase significantly. This is due to the fact that when random costs are used, there exists a lot of strategies with weight below the best path-strategy (see $\text{inc}_{S\text{-}PS}$ column). Hence the procedure first has to remove all these strategies from the candidate set before the best path-strategy can be found. However, when the best path-strategy is found, the increase in weight is small (see *inc* column). The situation is opposite when peak dependent costs are used here the increase between the first strategy and the first path-strategy in small and the increase in the weight for the first 100 path-strategies is high.

Procedure *K-BPSreopt* and *K-BPS_MB* was also tested on hypergraph class 12-17. The results are reported in Table 4.8. The columns $|\mathcal{E}_\pi|$, $|A_\pi|$, and $s = j$ are not shown since results similar to the one shown in Table 4.7 are obtained.

Class	ite_K	*reins*	CPU_K	ite_1	CPU_1	$\|\tau\|$	ite_K	*reins*	CPU_K	ite_1	CPU_1	$\|\tau\|$
MEC	*K-BPSreopt*						*K-BPS_MB*					
12	542	32	6	20	0.2	3	484	34	5	19	0.2	6
13	664	30	23	14	0.5	4	589	35	20	12	0.4	8
14	3732	36	375	194	20.1	4	3379	42	500	183	27.3	8
15	2297	38	24	205	2.2	2	1995	47	20	178	1.8	4
16	16336	39	529	4430	146.1	2	13896	50	424	3752	115.8	4
17	334272	39	27333	145971	11927.6	2	280451	51	27529	123048	12107.4	4
MMC	*K-BPSreopt*						*K-BPS_MB*					
12	350	25	6	5	0.1	3	316	25	5	5	0.1	7
13	549	25	29	20	1.1	4	498	29	26	18	1.0	8
14	5073	36	768	355	55.1	4	4604	41	710	338	55.3	8
15	951	12	10	122	1.3	2	826	13	9	107	1.1	4
16	5721	13	182	1721	54.8	2	4971	14	164	1523	50.3	4
17	709453	18	59904	360947	30414.8	2	584794	23	58898	297955	29825.2	4

Table 4.8: Results for finding the $K = 100$ best path-strategies using procedure *K-BPSreopt* and *K-BPS_MB* (no waiting).

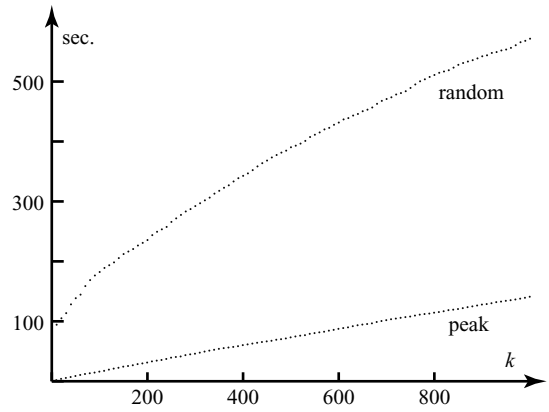


Figure 4.16: CPU time for increasing k for a hypergraph using random and peak dependent costs.

Clearly procedure *K-BPSreopt* and *K-BPS_MB* outperform procedure *K-BPS*. Moreover, procedure *K-BPS_MB* seems to have a slightly better CPU running time than procedure *K-BPSreopt*. This may be a result of the procedure obtaining a more tight lower bound when considering a subhypergraph in the branching operation. Note that the branching tree is approximately doubled in size when using procedure *K-BPS_MB*. However, since we create more constrained subgraphs in the branching operation, we obtain more subgraphs where no *o-d* paths exist.

Class	ite_K	reins	CPU_K	ite_1	CPU_1	$\lvert\tau\rvert$	$\lvert\mathcal{E}_\pi\rvert$	$\lvert A_\pi\rvert$	$s=1$	$s=2$	$s=3$	$s=4$	inc	$\text{inc}_{S\text{-}PS}$
MEC														
18	386	85	8.07	13	0.29	6	136	36	83	13	3	1	46	3
19	411	104	8.58	13	0.28	6	171	36	81	13	4	1	49	5
20	411	104	8.60	13	0.29	6	194	36	81	13	4	1	49	5
21	520	108	5.67	29	0.33	4	227	83	73	23	4	1	23	17
22	1476	600	15.53	112	1.17	4	234	117	58	25	15	2	18	33
23	1806	819	18.80	159	1.65	4	234	122	53	24	19	4	17	39
MMC														
18	245	28	5.32	4	0.11	7	119	19	87	10	3	0	42	1
19	292	52	6.22	5	0.12	7	190	19	86	12	2	0	47	3
20	292	52	6.27	5	0.12	7	221	19	86	12	2	0	47	3
21	350	29	4.03	18	0.20	5	219	23	78	20	2	0	25	12
22	709	99	7.73	75	0.82	4	252	30	64	30	5	0	13	23
23	748	93	8.31	79	0.87	4	252	30	64	30	6	0	14	23

Table 4.9: Results for finding the $K = 100$ best path-strategies corresponding to K different *o-d* paths in G using procedure *K-BPS_MB* (waiting allowed).

Finally, we tried to investigate the computational effort required by each generated path-strategy more carefully. To this aim we recorded the elapsed CPU time for every tenth generated path-strategy. Figure 4.16 on the preceding page shows the CPU time for finding k path-strategies of a hypergraph in class 13 and 16 using the same seed and procedure *K-BPS_MB*. The CPU time when using peak dependent costs seems to be linear in k and much lower than when using random costs. On the contrary, the CPU time when using random costs seems to follow a downward curved line and finding the first path-strategy takes much computational effort. As pointed out above, this is due to the fact that there exists a lot of strategies with weight below the best path-strategy. Afterwards the subgraphs in the candidate set are so constrained that, finding the k'th path-strategy, takes less time than finding the k-1'th path-strategy.

Waiting

Two problems is considered if waiting is allowed on hypergraph classes 18-23, namely finding the K best path-strategies corresponding to K different *o-d* paths in G and the problem of finding the K best path-strategies where the *o-d* paths in G do not have to be different.

The results for finding the $K = 100$ best path-strategies corresponding to K different *o-d* paths in G using procedure *K-BPS_MB* are reported in Table 4.9, while the results for finding the $K = 100$ best path-strategies using procedure *K-BPS_MB* are reported in Table 4.10. If we compare the results in the two tables, we see that the CPU time is smaller when we find the K best path strategies where the paths do not have to be different. This is due to the fact that we almost always find path-strategies corresponding to the same path (see the diff column). That is, we use Branching Operation 4.1.2 and

Class	ite_K	reins	diff	CPU_K	ite_1	CPU_1	$\lvert\tau\rvert$	$\lvert\mathcal{E}_\pi\rvert$	$\lvert A_\pi\rvert$	$s=1$	$s=2$	$s=3$	$s=4$	inc	$\text{inc}_{S\text{-}PS}$
MEC															
18	112	2	1	2.40	13	0.30	17	149	18	84	13	2	1	0	3
19	112	3	1	2.41	13	0.29	6	224	20	80	13	6	1	0	5
20	112	3	1	2.40	13	0.29	6	258	20	80	13	6	1	0	5
21	129	5	1	1.57	29	0.34	16	762	42	54	36	8	2	0	17
22	211	40	1	2.41	112	1.17	13	1008	60	37	30	27	6	0	33
23	258	64	1	2.88	159	1.65	9	1008	66	34	28	30	9	0	39
MEC															
18	103	0	1	2.18	4	0.12	55	125	18	84	12	4	0	0	1
19	104	1	1	2.22	5	0.13	68	196	18	84	13	3	0	2	3
20	111	2	4	2.40	5	0.13	92	279	18	85	12	3	0	8	3
21	118	1	1	1.42	18	0.21	17	391	34	63	32	5	0	1	12
22	174	10	1	2.06	75	0.84	18	387	42	48	40	10	2	0	23
23	178	6	1	2.08	79	0.86	18	400	44	47	43	10	0	0	23

Table 4.10: Results for finding the $K = 100$ best path-strategies using procedure *K-BPS_MB* (waiting allowed).

branch on hyperpaths instead of subpaths P_π. As a result, the increase in the weight becomes small again. If we consider the problem of finding the K minimum weight paths in G, we get more or less the same results as if waiting was not allowed. Note, however, that the increase in the weight between the first strategy and the first path-strategy falls under waiting, in particular when low waiting costs are used.

Further remarks

A few further remarks on the procedures under a priori route choice is in order. Consider the time-expanded hypergraph $\mathcal{H}$ and a node v in the topological network G. In the current implementation, the hyperarcs in $BS(v^t)$, $v^t \in \mathcal{V}_\mathcal{H}(v)$ are ordered in the same way. That is, if the hyperarc $e_a(t)$ corresponding to leaving node v along arc $a \in FS(v)$ at time t is ordered first in the backward star of a node $v^t \in \mathcal{V}_\mathcal{H}(v)$, then it is also ordered first in all nodes $v^{t'} \in \mathcal{V}_\mathcal{H}(v) \setminus \{v^t\}$. This is not relevant in the hypergraph classes on which we test. However, this may be relevant in some particular situations, e.g. if the cost of all the hyperarcs in $\mathcal{H}$ are the same. Then the order of hyperarcs becomes a rule for breaking ties and by picking e.g. the first hyperarc in the backward star of all the nodes in $\mathcal{V}_\mathcal{H}(v)$, we follow a single arc in the forward star of node v in G.

Another procedure was also tested under a priori route choice. The branching operations used under a priori route choice all branch on a subpath P_π in G. However, instead of branching on a subpath in G, we might use procedure *K-BSreopt*, i.e. branch on hyperpaths instead and then check if the strategy is a path-strategy using Corollary 4.2.1. Hence we can use the reoptimization techniques we have for hyperpaths which are faster. But a lot more branching tree nodes are generated using this approach. However, subhypergraphs can be skipped, if the following holds. Consider a subhypergraph $\mathcal{H}^{k,i}$ and

its end-tree $\eta^{k,i}$ when using Branching Operation 4.1.2 on the k'th hyperpath π^k. If the following holds for v, v' with $arc\left(p\left(v\right)\right), arc\left(p\left(v'\right)\right) \neq null$.

$$\exists v, v' \in I_\eta : v, v' \in \mathcal{V}_{\pi^k}\left(u\right) \text{ and } arc\left(p\left(v\right)\right) \neq arc\left(p\left(v'\right)\right)$$

then no hyperpaths in $\mathcal{H}^{k,i}$ can correspond to a path and we may skip $\mathcal{H}^{k,i}$ (and sub-hypergraphs $\mathcal{H}^{k,j}$, $j = 1, ..., i-1$). However, using this procedure leads to much higher CPU times than the ones reported in the tables.

4.3.4 Summary

This section presents a short summary of the main results of the computational tests under a priori and time-adaptive route choice.

First, consider time-adaptive route choice where the structure of the costs have a big impact on how path-like the K best strategies are. That is, high variation in the costs for different leaving times t when considering a given node u and leaving arc in G, results in that the best strategies are not very path-like. High variation in the costs can be obtained by using peak dependent costs and increasing the peak increase parameter ψ, using peak dependent costs and increasing the range r_ξ of the random perturbation or using random costs which makes the strategies most non path-like.

The best strategy can be found very fast. Moreover, procedure *K-BSreopt* outperforms procedure *K-BS*.

The performance of procedure *K-BSreopt* is affected by the size of $\mathcal{H}$, the size of the hyperpaths we branch on and the number of new branching tree nodes created. The CPU times of procedure *K-BSreopt* when considering the MEC and MMC criteria is more or less the same. The procedure performance is not affected significantly by allowing waiting.

The increase in the weight between the first strategy and the K'th strategy is negligible.

Second, consider a priori route choice. Here the structure of the costs may have a big impact on procedure performance. High variation in the costs for different leaving times t when considering a given node u and leaving arc in G, result in that the K best strategies are not very path-like. This will affect the procedures under a priori route choice, since, if strategies are not path-like, then the length of the path P_π used in the branching operation will be small. Hence when using random costs we may have to perform a lot of branching operations before a path-strategy is found. On the other hand if the strategies are more path-like then the length of P_π will be long and we will find a path-strategy faster. As a result using random costs will have a very negative impact on procedure performance.

Procedure *K-BPSreopt* and *K-BPS_MB* outperform procedure *K-BPS* and finding the K best path-strategies can be done relatively fast on "realistic" STD networks, i.e. STD networks using peak dependent costs.

If comparing procedure *K-BPSreopt* and *K-BPS_MB*, procedure *K-BPS_MB* seems to be slightly better. However, more memory has to be used in procedure *K-BPS_MB*.

The increase in the weight between the first path-strategy and the K'th path-strategy is much larger than under time-adaptive route choice if we find path-strategies corresponding to different *o-d* paths in G. If waiting is allowed and the *o-d* paths do not have to be different the K best path-strategies often will correspond to the same path in G resulting in a small increase between the first path-strategy and the K'th path-strategy.

Finally, if we compare the procedures under time-adaptive route choice and a priori route choice we see that finding the K best path-strategies is harder than finding the K best strategies. Moreover, the structure of the costs have a much larger effect on procedure performance under a priori route choice. Finally, the branching tree size is much smaller under a priori route choice, i.e. less memory has to be used.

Chapter 5 Bicriterion route choice in STD networks

One of the most classical problems encountered in the analysis of networks is the shortest path problem. Traditionally the shortest path problem was a single objective problem with the objective being to minimize total distance or travel time. Nevertheless, due to the multiobjective nature of many transportation and routing problems, a single objective function is not sufficient to completely characterize most real-life problems. In a road network for instance, two parameters, time and cost, can be assigned to each arc. Clearly, often the fastest path may be too costly or the cheapest path may be too long. Therefore the decision maker must choose a solution among the paths for which it is not possible to find a different path such that time or cost is improved without getting a worse cost or time, respectively (*efficient path*).

The problem of finding all efficient paths is denoted the bicriterion shortest path (*bi-SP*) problem and has generated wide interest in multicriterion linear integer programming; for an overview on multicriterion optimization see Ehrgott [23] or Ehrgott and Gandibleux [24].

An efficient path P defines a *nondominated point* $(W_1(P), W_2(P))$ in the criterion space, where $W_j(P)$ denotes the length/weight of the path with respect to the j'th criterion. Due to the discrete nature of the bi-SP problem, the set of nondominated points is not convex. We partition them into two sets. Basically, if a point corresponding to P lies on the boundary of the convex hull of all the nondominated points, it is a supported nondominated point, otherwise it is an unsupported nondominated point.

Hansen [39] introduced the bi-SP problem and other related bicriterion shortest path problems and a few years later Climaco and Martins [16] presented experimental results for a simple procedure solving the bi-SP problem using K shortest paths. The method seems to be slow, since there are too many paths to search, see Mote, Murthy, and Olson [63]. Garey and Johnson [33] showed that the bi-SP problem is $\mathcal{NP}$-hard. A variety of algorithms has been developed to solve bi-SP based on dynamic programming, see e.g. Sancho [82] and Henig [40] and label setting/correcting methods, see e.g. Martins [55] and Tung and Chew [89]. Here a set of labels is maintained in each node v in the network where each label contains the weights of an efficient path to node v. The label correcting approach uses the well-known fact that all efficient paths pass through efficient subpaths. Therefore a label only has to be maintained for each efficient path. The

two-phase approach has also been used on the bi-SP problem, see Mote et al. [63]. The two-phase approach is a general method of finding efficient solutions which have been used to solve bi-SP and other biobjective combinatorial problems, see e.g. Ulungu and Teghem [90] and Visée, Teghem, Pirlot, and Ulungu [91]. More recent *interactive* methods, where only a subset of the nondominated points is found, have been considered by Coutinho-Rodrigues et al. [20] and Current, ReVelle, and Cohen [21]. Here a two-phase approach is used where the first phase finds supported nondominated points by solving shortest path problems and the second phase finds unsupported nondominated points by using a K shortest paths procedure. Computational experiments comparing different methods of the bi-SP problem have been given by Huarng, Pulat, and Shih [44], Brambaugh-Smith and Shier [10] and Skriver and Andersen [85]. For an annotated bibliography of the bi-SP problem including more than forty references, we refer to Ehrgott and Gandibleux [24].

It is obvious that problems concerning bicriterion route choice in STD networks are relevant. For instance, if routing hazardous materials in an STD network, two criteria may be considered, namely expected risk and expected travel time, i.e. the first objective is MEC where the cost corresponds to risk and the second objective is MET (see Section 3.2). Note that other criteria may be used, e.g. instead of minimizing expected risk the decision maker may be more risk averse and wish to minimize maximum risk instead (MMC). Therefore bicriterion route choice problems in STD networks have a richer structure than bi-SP, in the sense that the two criteria may correspond to time as well as cost and the purpose may be to minimize the expected as well as the maximum possible time or cost.

In this chapter we consider bicriterion route choice problems in STD networks under a priori and time-adaptive route choice. A two-phase approach is used to find efficient strategies. The two-phase approach will be described in Section 5.2.

Under time-adaptive route choice, the problem consists in finding the set of efficient strategies between an origin and destination node when leaving the origin at time zero. To the author's knowledge, no one has yet considered this problem. Since a strategy S_{o0} corresponds to a hyperpath in the time-expanded hypergraph $\mathcal{H}$, finding the set of efficient strategies corresponds to finding the set of efficient hyperpaths in the time-expanded hypergraph $\mathcal{H}$, all having source s and target t corresponding to leaving the origin at time zero. Procedures solving the problem are presented in Section 5.4. Unfortunately, computational results show that the number of efficient strategies may grow exponentially in hypergraph size. Therefore finding all efficient strategies may be impossible in practice. As a result procedures finding an approximation of the set of efficient strategies are developed.

The problem under a priori route choice consists in finding the set of efficient path-strategies where each path-strategy defines a path followed between the origin and destination node when leaving the origin at time zero. Note that, if no waiting is allowed in the nodes in G, then, according to Corollary 4.2.3, a path in G corresponds to a unique path-strategy. Hence each path-strategy in the efficient set defines a unique path in G. However, if waiting is allowed, then different path-strategies may correspond to the same path. Only one paper has considered the problem of finding efficient strategies under a priori route choice. Miller-Hooks and Mahmassani [60] consider the problem on STD networks where no waiting is allowed with the first objective being MET and the second objective being MEC. A labelling procedure is presented which guarantees that all the efficient paths can be obtained; however, in practice the procedure is too slow. Therefore a heuristic procedure to produce a subset of the efficient paths is suggested. The proce-

dures presented in this chapter will not use a labelling approach to find efficient strategies, instead a two-phase approach is used. Since the bi-SP problem is $\mathcal{NP}$-hard and bi-SP is a special case of the problems we consider here, we have that they are $\mathcal{NP}$-hard as well.

5.1 Basic definitions

We start by introducing some basic definitions extended from directed graphs to directed hypergraphs. We follow the terminology given in Skriver [84].

Given a hypergraph $\mathcal{H}$, assume that each hyperarc e is assigned two nonnegative real weights $w_1(e)$ and $w_2(e)$. Furthermore, let $W_j(\pi)$, $j = 1, 2$, denote the weight of hyperpath π using weights $w_i(e)$.

Definition 5.1.1 Let Π denote the set of s-t hyperpaths in $\mathcal{H}$. A hyperpath $\pi \in \Pi$ is *efficient* if and only if

$$\nexists \tilde{\pi} \in \Pi : W_1(\tilde{\pi}) \leq W_1(\pi) \text{ and } W_2(\tilde{\pi}) \leq W_2(\pi)$$

with at least one strict inequality; otherwise π is *inefficient*.

Efficient hyperpaths are defined in the decision space Π and correspond to points in the *criterion space*:

$$\mathcal{W} = \left\{W(\pi) \in \mathbb{R}^2 \mid \pi \in \Pi\right\}$$

where $W(\pi) \in \mathbb{R}^2$ is the vector with components $W_1(\pi)$ and $W_2(\pi)$.

Definition 5.1.2 A point $W(\pi) \in \mathcal{W}$ is a *nondominated* criterion point if and only if π is an efficient hyperpath. Otherwise $W(\pi)$ is a *dominated* criterion point.

Let us define

$$\Pi_{\mathit{eff}} = \{\pi \in \Pi \mid \pi \text{ is efficient}\}, \quad \mathcal{W}_{\mathit{eff}} = \left\{W(\pi) \in \mathbb{R}^2 \mid \pi \in \Pi_{\mathit{eff}}\right\}$$

Finding the set of efficient hyperpaths Π_{eff}, implies that we find the set of nondominated criterion points $\mathcal{W}_{\mathit{eff}}$. However, two efficient hyperpaths may correspond to the same nondominated point in $\mathcal{W}_{\mathit{eff}}$. Therefore we do not need to find all efficient hyperpaths for finding all nondominated points but only one efficient hyperpath for each nondominated point. Most papers in the literature consider finding Π_{eff} and $\mathcal{W}_{\mathit{eff}}$ as equivalent and finds $\mathcal{W}_{\mathit{eff}}$. However, often the solution procedures only need to be changed slightly to find Π_{eff}. In this chapter we consider the problem of finding Π_{eff} except in a few special cases.

The criterion points can be partitioned into two kinds, namely supported and unsupported. The supported ones can be further subdivided into extreme and nonextreme. To this aim, let us define the following set

$$\mathcal{W}^{\geq} = \mathit{conv}(\mathcal{W}_{\mathit{eff}}) \oplus \left\{\mathbf{w} \in \mathbb{R}^2 \mid \mathbf{w} \geq 0\right\};$$

where $\oplus$ as usual denotes direct sum, and $\mathit{conv}(\mathcal{W}_{\mathit{eff}})$ denotes the convex hull of $\mathcal{W}_{\mathit{eff}}$.

Definition 5.1.3 $W(\pi) \in \mathcal{W}$ is a *supported* nondominated criterion point if $W(\pi)$ is on the boundary of $\mathcal{W}^{\geq}$. Otherwise $W(\pi)$ is an *unsupported* point.

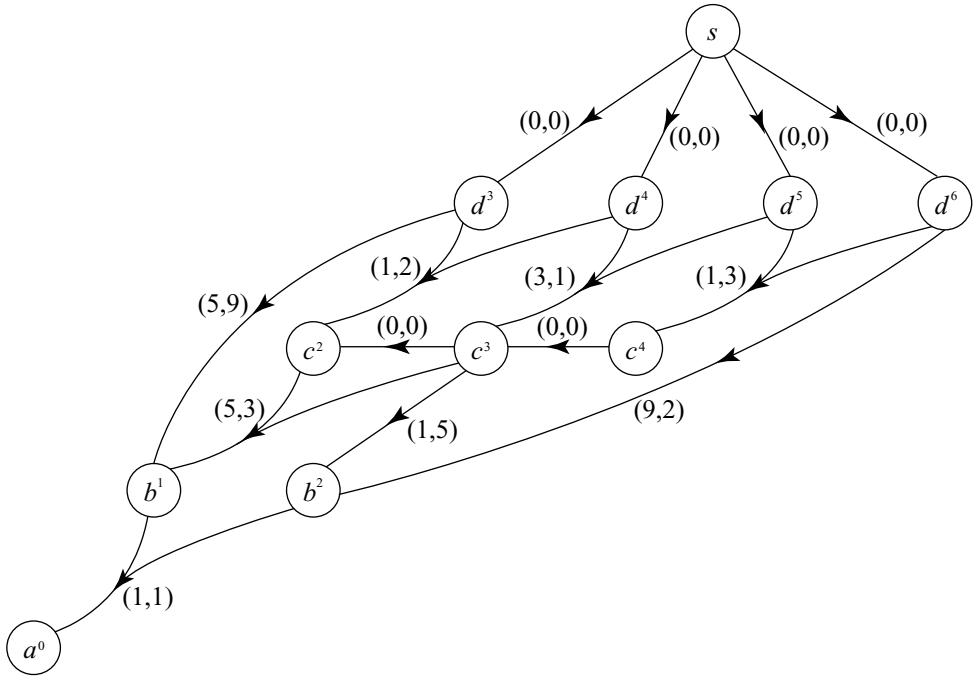


Figure 5.1: The time-expanded hypergraph.

Definition 5.1.4 A supported point $W(\pi)$ is an *extreme* if $W(\pi)$ is an extreme point of $\mathcal{W}^{\geq}$. Otherwise $W(\pi)$ is a *nonextreme* point.

It is well-known that unsupported nondominated points (in fact, all vectors in $\mathcal{W}$) are dominated by a convex combination of extreme supported points (Steuer [86]). Moreover, a set of nondominated points $\Phi = \left\{W^1, W^2, \ldots, W^r\right\} \subseteq \mathbb{R}^2$ can be ordered such that:

$$W_1^1 < W_1^2 < ... < W_1^r, \qquad W_2^1 > W_2^2 > ... > W_2^r \tag{5.1}$$

We call Φ an *ordered nondominated set*. We denote by the *frontier*, the ordered nondominated set of extreme supported points in $\mathcal{W}$. The following definitions introduce the concepts of ε-domination and ε-approximation. The definitions below follow the terminology given by Warburton [92].

Definition 5.1.5 A point (W_1, W_2) *ε-dominates* point $(\hat{W}_1, \hat{W}_2)$ if

$$\hat{W}_1 \geq (1-\varepsilon)\, W_1, \qquad \hat{W}_2 \geq (1-\varepsilon)\, W_2$$

Definition 5.1.6 A set Φ_1 is an *ε-approximation* of another nondominated set Φ_2, if for each point $\hat{W} \in \Phi_2$, there exists $W \in \Phi_1$ such that W ε-dominates $\hat{W}$.

Example 4 Given the topological network G and time-expanded hypergraph of Example 3 on page 26, assume that two costs $c_i\,(u, v, t)$, $i = 1, 2$, are given for each leaving

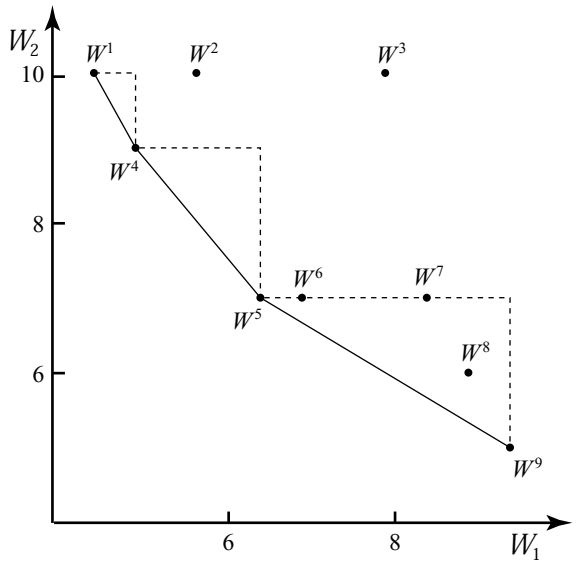


Figure 5.2: The criterion space of the hyperpaths in $\mathcal{H}$.

time t from node u along arc (u, v). Moreover, we assume that the penalty costs and waiting costs are equal to zero. The time-expanded hypergraph $\mathcal{H}$ and the costs are given in Figure 5.1. Assume that we are interested in finding the set of nondominated points under time-adaptive and a priori route choice when the first and second criterion is MEC and MMC, respectively.

First, consider the problem of finding efficient strategies under time-adaptive route choice. According to Theorem 3.3.1 the problem consists in finding efficient s-a_0 hyperpaths π with weights $(W_1(\pi), W_2(\pi))$ where $W_1(\pi)$ is the weight of hyperpath π using weights $w_1(e)$ and the mean weighting function, and where $W_2(\pi)$ is the weight of hyperpath π using weights $w_2(e)$ and the distance weighting function.

Hypergraph $\mathcal{H}$ contains eleven s-a_0 hyperpaths corresponding to nine points in the criterion space. The points are given by: $W^1 = (4.5, 10)$, $W^2 = (5.5, 10)$, $W^3 = (8, 10)$, $W^4 = (5, 9)$, $W^5 = (6.5, 7)$, $W^6 = (7, 7)$, $W^7 = (8.5, 7)$, $W^8 = (9, 6)$ and $W^9 = (9.5, 5)$. These points are illustrated in Figure 5.2; the frontier consists of the four points W^1, W^4, W^5 and W^9; solid lines joining points in the frontier belong to the boundary of $W^{\geq}$. Note that the frontier defines three triangles, shown with dashed lines, where it may be possible to find unsupported nondominated points such as W^8. Points which do not lie inside the triangles such as W^3 and W^6 are dominated.

Next, consider the problem of finding efficient strategies under a priori route choice, i.e. we are interested in finding efficient hyperpaths corresponding to a path-strategy. Hypergraph $\mathcal{H}$ contains only five hyperpaths defining a path-strategy which correspond to points W^3, W^4, W^5, W^6 and W^8. That is, the set of nondominated criterion points is $\{W^4, W^5, W^8\}$ which all are extreme nondominated points under a priori route choice.■

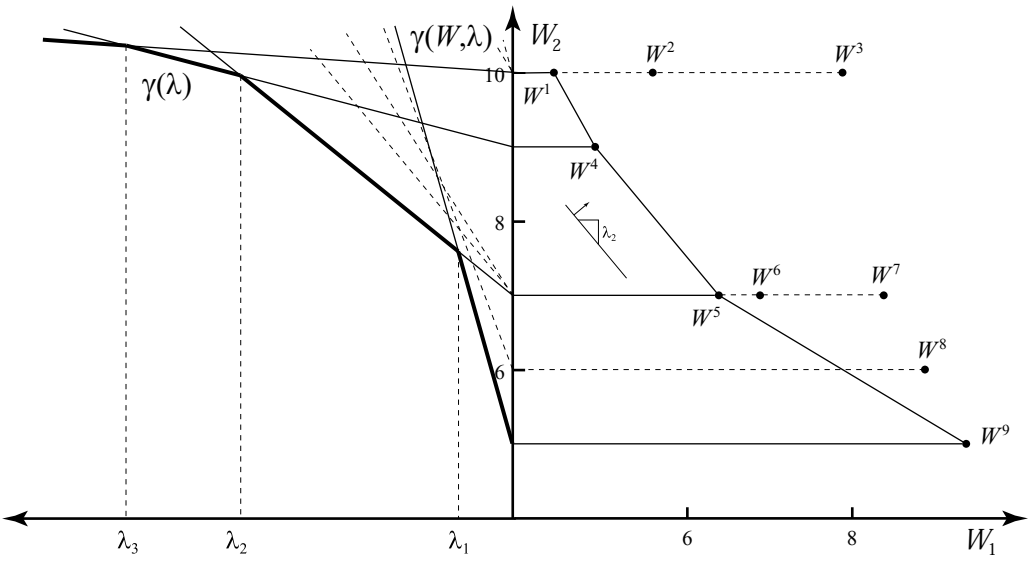


Figure 5.3: The criterion space and its corresponding parametric space.

5.2 The two-phase approach

The two-phase approach is a general method for solving bicriterion combinatorial problems. It cannot be used for more than two criteria; since phase one may fail in finding all extreme nondominated points.

As the name suggests, the two-phase method splits the search of nondominated points into two phases. In phase one the supported extreme nondominated points are found. These extreme points define the triangles in which unsupported nondominated points may be found. Phase two proceeds to search the triangles one at a time. This is done parametrically. The approach is illustrated by the set of criterion points shown in Figure 5.2 representing the criterion weights for the s-a_0 hyperpaths in Figure 5.1. Let $\gamma : (\mathcal{W}, \mathbb{R}_+) \to \mathbb{R}_+$ denote the *parametric weight* of a hyperpath π.

$$\gamma\left(W\left(\pi\right), \lambda\right) = W_1(\pi)\lambda + W_2\left(\pi\right). \tag{5.2}$$

Given $\lambda > 0$, a *minimum parametric weight hyperpath* $\pi\left(\lambda\right)$ (there may be many), is a hyperpath with minimal parametric weight denoted $\gamma(\lambda)$. The criterion space and its corresponding parametric space are shown in Figure 5.3. For a given hyperpath $\pi \in \Pi$, each point $W\left(\pi\right)$ corresponds to a line with slope $W_1\left(\pi\right)$ and intersection $W_2\left(\pi\right)$ in the parametric space. Given a fixed $\lambda > 0$, we have a line in the parametric space defined by some $\pi \in \Pi$ which minimizes $\gamma(W\left(\pi\right), \lambda)$, see Figure 5.3. Moreover, the lower envelope of the lines in the parametric space defines $\gamma\left(\lambda\right)$, which is a non-decreasing piecewise linear function with break points λ_i. Note that each breakpoint λ_i corresponds to a value of λ where two adjacent supported nondominated points have the same minimal parametric weight. For instance for $\lambda = \lambda_2$ we have that W^4 and W^5 have the same minimal parametric weight, i.e. finding a minimal parametric weight hyperpath $\pi\left(\lambda_2\right)$

```
1  procedure phaseOne(H)
2      π(M) := SHTayclic(H, M); add(Φ, W(π(M)));
3      π(ε) := SHTayclic(H, ε); add(Φ, W(π(ε)));
4      if (W(π(M)) = W(π(ε))) then STOP (only one nondominated point)
5      W+ = W(π(M)); W− = W(π(ε));
6      while (W+ ≠ W(π(ε))) do
7          λ := |(W2− − W2+)/(W1− − W1+)|;
8          inc := false;
9          π(λ) := SHTayclic(H, λ);
10         if (γ(λ) < γ(W+, λ)) then add(Φ, W(π(λ));
11         else inc := true;
12         if (inc) then W+ := W−;
13         W− := next(Φ, W+);
14     end while
15 end procedure
```

Figure 5.4: Phase one - Finding extreme nondominated points.

corresponds to searching in the direction of the normal to the line shown in Figure 5.3. It is obvious that each piece of $\gamma(\lambda)$ defines an extreme nondominated point. Hence the extreme nondominated points can be found by finding a minimal parametric weight hyperpath $\pi(\lambda)$ for different values of λ. This is done by using a NISE* like algorithm (see Cohen [17]) shown in Figure 5.4. Procedure *phaseOne* uses the following subprocedures

SHTacyclic$(\mathcal{H}, \lambda)$: Finds the minimum parametric weight hyperpath $\pi(\lambda)$ for a given λ.

add(Φ, W): A general function for adding a point to the ordered nondominated set Φ. If W not is dominated by a point in Φ then W is added to Φ and points now dominated by W are removed. Moreover, W is added to Φ so that afterwards Φ is ordered as in (5.1).

next(Φ, W): Return the point following W in Φ, i.e. if $W = W^i$ then W^{i+1} is returned.

The procedure first finds the *upper/left* and the *lower/right* point (W^1 and W^9 in Figure 5.3). The upper/left point is the nondominated point which has minimum weight using the second criterion when weight one is fixed to its minimum weight. Similarly, the lower/right point is the nondominated point which has minimum weight using the first criterion when weight two is fixed to its minimum weight. In general the upper/left point can be found by finding $\pi(\lambda)$ with λ very big (line 2, M big) and the lower/right point can be found by finding $\pi(\lambda)$ with λ close to zero (line 3, ε small).

Given two nondominated points W^+ and W^-, we now calculate the search direction λ defined by the slope of the line between the points and find the minimum parametric weight hyperpath $\pi(\lambda)$. If $W(\pi(\lambda))$ corresponds to a new extreme nondominated point, then the parametric weight $\gamma(\lambda)$ of $\pi(\lambda)$ must be below the parametric weight of W^+ and

*Non-inferior set estimation.

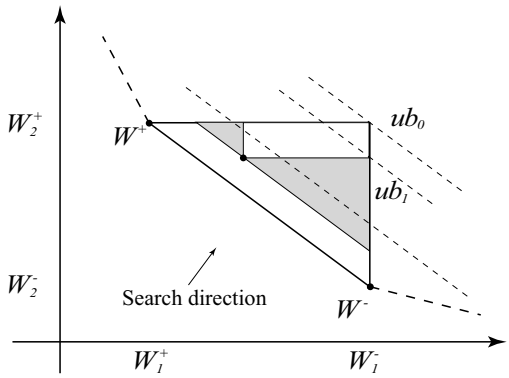


Figure 5.5: A triangle defined by W^+ and W^-.

W^- (line 10). The points $W^+, W(\pi(\lambda))$ and W^- then define two new search directions and the **while** step is repeated on the points W^+ and $W(\pi(\lambda))$. Otherwise no new extreme nondominated point has been found and we proceed with the two next points in Φ. The procedure stops when no more new extreme nondominated points can be found.

Since the number of hyperpaths in Π is finite, we have that the number of lines defining $\gamma(\lambda)$ in the parametric space is finite and hence procedure *phaseOne* will stop in a finite number of steps. However, the number of extreme nondominated points may increase exponentially in hypergraph size. In this case an approximation of the extreme nondominated points may be found, as we shall see in Section 5.4

Finally, observe that if the search direction is defined by the slope between two adjacent extreme nondominated points, it may happen that the minimum parametric weight hyperpath $\pi(\lambda)$ corresponds to a supported nonextreme nondominated point. This point is excluded on line 10; however, often including supported nonextreme nondominated points may reduce the space to search by phase two. Procedure *phaseOne* can easily be modified so that nonextreme nondominated points are included in Φ by changing the less than sign on line 10 to a less than or equal sign.

Since there may exist unsupported nondominated criterion points, it is not in general possible to find all nondominated points during the first phase. This can be seen in Figure 5.3 where unsupported nondominated points inside the triangles, such as W^8, correspond to a dashed line lying above $\gamma(\lambda)$. These points are found in phase two which searches each triangle defined by the set of extreme nondominated points found in phase one. Consider the triangle defined by the extreme nondominated points W^+ and W^- (see Figure 5.5). The second phase searches each triangle using a K best strategy procedure in the direction λ defined by the slope between the two points defining the triangle. The procedure stops when an upper bound has been reached. At start the upper bound is $ub_0 = W_1^-\lambda + W_2^+$. However, when a new unsupported nondominated point is found inside the triangle, the upper bound is updated to ub_1 (see Figure 5.5). Note that the procedure may find points outside the triangle because all points with parametric weight below ub_1 are found.

5.3 Label correcting algorithms

In this section we point out some disadvantages of using label correcting algorithms for finding efficient hyperpaths in directed hypergraphs. Label correcting algorithms have proven successful in solving the bicriterion shortest path problem see e.g. Skriver and Andersen [85]. Basically, label correcting algorithms maintain a set of labels in each node v in the network where each label contains the weights of a path to node v. The labels of each node in the network are then updated until all nondominated points have been found. The method uses the well-known fact that all efficient paths pass through efficient subpaths. Hence only labels corresponding to efficient paths have to be maintained in each node. Note that the number of efficient paths is always larger than the number of nondominated points because there may be more than one efficient path corresponding to a nondominated point. Hence one may argue that label correcting algorithms are faster in solving the bicriterion shortest path problem than algorithms using the two-phase approach, since label correction algorithms only find one efficient path for each nondominated point, while the two-phase method has to find all efficient paths when searching a triangle. However, label correcting algorithms also have some disadvantages.

First, label correcting algorithms find all nondominated points and do not distinguish between supported and unsupported nondominated points. Therefore, if the number of nondominated points is very large, we cannot use interactive methods as in the two-phase approach where the supported extreme nondominated points first may be found and next the triangles of interest can be searched for further unsupported nondominated points.

Second, as will be pointed out in the following sections, the set of nondominated points may be huge. Therefore an approximation of the nondominated set has to be found. The two-phase approach seems to be more suitable for this.

Finally, the well-known fact that all efficient paths pass through efficient subpaths cannot be extended to hyperpaths as can be seen in the following example.

Example 5 Consider the time-expanded hypergraph $\mathcal{H}$ shown in Figure 5.6. Hypergraph $\mathcal{H}$ is similar to the time-expanded hypergraph in Example 4. However, here the costs on arc $e_{bd}(2)$ is changed from $(9,2)$ to $(7,7)$. Assume that we are interested in finding the set of efficient hyperpaths, i.e. efficient strategies under time-adaptive route choice. Let the first and second criterion be MMC, i.e. two distance weighting functions are used.

The set of nondominated points is $\Phi = \{(6,10),(7,9),(8,8),(9,7)\}$. The hyperpath corresponding to point $(8,8)$ is shown in bold in Figure 5.6. Note that the subhyperpath to node b_2 is dominated by another s-b_2 hyperpath with weight $(4,6)$. That is, the efficient hyperpath corresponding to nondominated point $(8,8)$ contains an inefficient subhyperpath. Furthermore, there does not exist another efficient s-t hyperpath with an efficient s-b_2 subhyperpath yielding point $(8,8)$. ■

The above example shows that we cannot extend label correcting algorithms to hypergraphs, since, by maintaining only labels corresponding to efficient hyperpaths in each node, we may not find all nondominated points. Similar examples can be constructed if other criteria are considered.

Another problem also arises if a label correcting algorithm is applied to the time-expanded hypergraph in Figure 5.6. Assume that both criteria is MEC, i.e. two mean weighting functions are used. Then the label set LS in node b^1 and b^2 will contain the

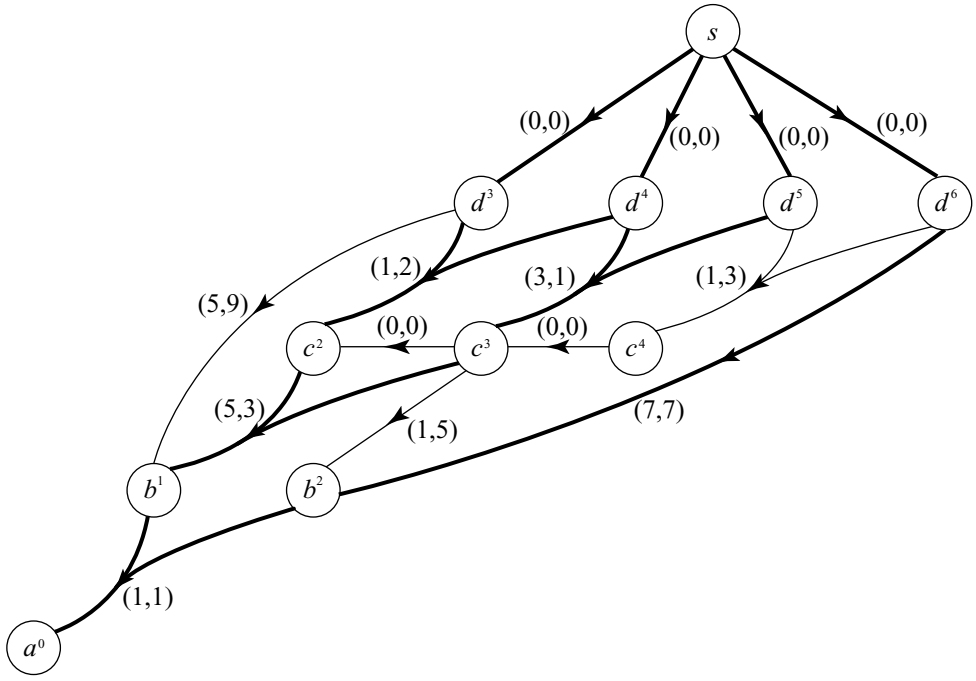


Figure 5.6: A time-expanded hypergraph $\mathcal{H}$.

following weights of efficient hyperpaths.

$$LS\left(b^1\right) = \{(5,9),(6,5.5),(7,4.5),(8,4)\}$$
$$LS\left(b^2\right) = \{(2,8),(4,6)\}$$

If we combine the weights in $L\left(b^1\right)$ and $L\left(b^2\right)$ and remove the dominated points in order to find the label set in node a^0, we get

$$LS\left(a^0\right) = \{(4.5,9.5),(5,7.75),(5.5,7.25),(6,6.75),(6.5,6.25),(7,6)\}$$

However, points $(5.5, 7.25)$ and $(6, 6.75)$ do not correspond to the weight of an s-a^0 hyperpath, i.e. they are not nondominated points. The reason is that here we assumed that the hypergraph containing hyperarc $e_{ab}(0)$ and two efficient hyperpaths from node s to node b^1 and b^2 is a hyperpath. This is not always true. Hence each label in a node must not only contain the weights of the hyperpath but also the hyperpath such that we can check that we still have a hyperpath when combining subhyperpaths. It is obvious that checking this may be time consuming. Moreover, much more memory has to be used for storing labels.

Due to the reasons pointed out above, the two-phase approach will be used to solve the problems in the following sections.

5.4 Finding the set of efficient strategies under time-adaptive route choice

In this section the problem of finding efficient strategies under time-adaptive route choice is considered. The problem consists in finding the set of efficient strategies between an origin and a destination node when leaving the origin at time zero. The criteria given in Section 3.2 are used, i.e. we consider expectation criteria where the goal is to minimize the expected travel time (MET) or cost (MEC) or min-max criteria where the goal is to minimize the maximum possible travel time (MMT) or cost (MMC).

Since a strategy between an origin and a destination node corresponds to a hyperpath in the time-expanded hypergraph $\mathcal{H}$, finding the set of efficient strategies corresponds to finding the set of efficient hyperpaths Π_{eff} in the time-expanded hypergraph $\mathcal{H}$, all having source s and target node t corresponding to leaving the origin at time zero.

Recall that, according to Theorem 3.3.1, both MEC and MET reduce to finding a minimum weight hyperpath using the mean weighting function. Similarly MMC and MMT reduce to finding a minimum weight hyperpath using the distance weighting function. Hence time and cost can be treated in a uniform way and we only have to focus on whether we are minimizing expectation criteria (MEC and MEC) or min-max criteria (MMC and MMT).

Three cases arise: if both criteria are expectation criteria, the resulting problem becomes finding efficient hyperpaths using two mean weighting functions. If both criteria are min-max criteria, we have to find efficient hyperpaths using two distance weighting functions. Finally, if one of the criteria is expectation and the other criterion is min-max, we have to find efficient hyperpaths using a mean weighting function for the expectation criterion and a distance weighting function for the min-max criterion. Each case will be treated separately, since the problem of finding a hyperpath with minimal parametric weight turns out to be harder in some cases than others.

Recall that for finding efficient strategies using the two-phase approach requires that we can solve minimum weight hyperpath and K minimum weight hyperpaths problems with respect to the parametric weight (5.2), which is a linear combination of the two criteria. It is well-known that, as long as directed graphs are considered, the following holds

Theorem 5.4.1 *Let $G = (N, A)$ be a directed graph where each arc $a \in A$ carries two weights $(w_1(a), w_2(a))$. Given a search direction defined by λ, the shortest parametric path corresponding to a supported nondominated point is the shortest path using weights $w_\lambda(a) = w_1(a)\lambda + w_2(a)$.*

That is, we can find a path with minimal parametric weight by just solving a shortest path problem on G with modified weights $w_\lambda(a)$. This is not necessarily the case for directed hypergraphs. In the following let $\mathcal{H}_\lambda = (\mathcal{V}, \mathcal{E})$ denote the hypergraph in which each hyperarc $e \in \mathcal{E}$ is assigned weight $w_\lambda(e) = w_1(e)\lambda + w_2(e)$.

5.4.1 Expectation criteria

Assume that both criteria are expectation criteria, i.e. we have to find efficient hyperpaths using two mean weighting functions. Remarkably, Theorem 5.4.1 can be extended to

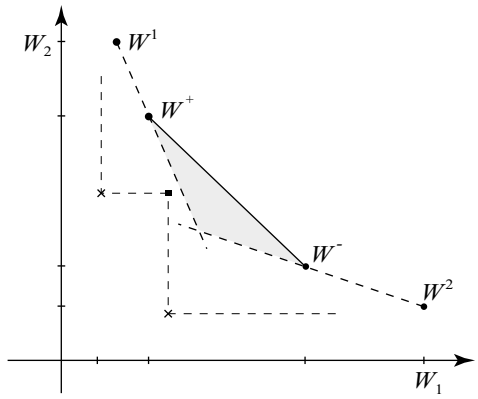


Figure 5.7: Using ε-dominance in the first phase.

hypergraphs when two mean weighting functions are considered.

Theorem 5.4.2 *Let $W_\lambda(\pi)$ denote the weight of a hyperpath π in $\mathcal{H}_\lambda$ using the mean weighting function. For every $\lambda > 0$ and for every $\pi \in \Pi$, we have that $W_\lambda(\pi) = \gamma(W(\pi), \lambda)$.*

Proof It suffices to write the weight $W_\lambda(\pi)$ of the s-t hyperpath π according to (5.2):

$$\begin{aligned} W_\lambda(\pi) &= \sum_{u \in \mathcal{V}_\pi \setminus \{s\}} f^\pi(u) w_\lambda(p(u)) \\ &= \sum_{u \in \mathcal{V}_\pi \setminus \{s\}} (f^\pi(u) w_1(p(u))\lambda) + (f^\pi(u) w_2(p(u))) \\ &= \lambda \sum_{u \in \mathcal{V}_\pi \setminus \{s\}} f^\pi(u) w_1(p(u)) + \sum_{u \in \mathcal{V}_\pi \setminus \{s\}} f^\pi(u) w_2(p(u)) \\ &= \gamma(W(\pi), \lambda). \end{aligned}$$

■

Theorem 5.4.2 provides us with the following corollary

Corollary 5.4.1 *Finding the hyperpath $\pi(\lambda)$ with minimal parametric weight $\gamma(\lambda)$ reduces to finding a minimum weight hyperpath in $\mathcal{H}_\lambda$.*

Phase one: Finding the frontier

Due to Corollary 5.4.1 the frontier can be found using procedure *phaseOne* shown in Figure 5.4 where subprocedure *SHTacyclic* simply finds the minimum weight hyperpath in $\mathcal{H}_\lambda$. However, as pointed out in Section 4.3, there may often be a lot of hyperpaths with weight close to each other. Hence there may be a large number of extreme nondominated points, resulting in high CPU times for finding the frontier. In some cases, we may be

satisfied with an ε-approximation of the true frontier. Let $\Phi = \left\{W^1, W^+, W^-, W^2\right\}$ denote an ordered nondominated set of four extreme nondominated points found during phase one, as shown in Figure 5.7. Note that any new extreme nondominated points between W^+ and W^- must belong to the shaded area in Figure 5.7.

Lemma 5.4.1 *Given $\Phi = \left\{W^1, W^+, W^-, W^2\right\}$, each extreme nondominated point between W^+ and W^-, i.e. inside the shaded area shown in Figure 5.7, is ε-dominated by either W^+ or W^- if*

$$\begin{array}{l}(1-\varepsilon)\, W_1^- \lambda_1 + (1-\varepsilon)\, W_2^+ \leq \gamma\left(W^1, \lambda_1\right) \quad or \\ (1-\varepsilon)\, W_1^- \lambda_2 + (1-\varepsilon)\, W_2^+ \leq \gamma\left(W^2, \lambda_2\right)\end{array} \tag{5.3}$$

where λ_1 denotes the slope defined by W^1 and W^+ and λ_2 the slope defined by W^2 and W^-.

Proof Given ε, W^+ and W^- define two points $(1-\varepsilon)\, W^+$ and $(1-\varepsilon)\, W^-$, shown with crosses in Figure 5.7. According to Definition 5.1.5, all points in the shaded area are ε-dominated if $(1-\varepsilon)\, W^+$ and $(1-\varepsilon)\, W^-$ dominate the shaded area. That is, the point $\left((1-\varepsilon)\, W_1^-, (1-\varepsilon)\, W_2^+\right)$, shown with a square in Figure 5.7, must either be below the line containing W^1 and W^+ or below the line containing W^2 and W^-. Using λ_1 and λ_2, we get the conditions in (5.3). ■

Procedure *phaseOne* in Figure 5.4 can now be modified to find an ε-approximation of the true frontier, denoted procedure *phaseOne*(ε). Simply in the beginning of the **while** loop, check whether (5.3) holds. If not, continue as before; otherwise set $inc = true$ and go to line 12. Due to Lemma 5.4.1 and the fact that, when a new nondominated point is added to Φ in procedure *phaseOne*, it is an extreme nondominated point, we have the following theorem

Theorem 5.4.3 *Procedure phaseOne(ε) finds an ε-approximation of the frontier in a finite number of steps containing a subset of the extreme nondominated points.*

Phase two: Looking into the triangles

After phase one an ordered nondominated set $\Phi = \left\{W^1, W^2, ..., W^r\right\}$ containing extreme nondominated points has been found. Note that Φ might be an approximation of the true frontier. Φ gives rise to a set of $r-1$ triangles in which further unsupported nondominated points may be found in phase two. Each triangle is searched independently, thus some triangles may be ignored, e.g. if an interactive approach is adopted, see e.g. Current et al. [21].

Assume that we consider the triangle defined by W^+ and W^- in Φ. Since Corollary 5.4.1 holds W^+ and W^- are extreme nondominated points of the frontier. Moreover, if (5.3) holds then nondominated points inside the triangle is ε-dominated by W^+ or W^-, and the triangle can be ignored if an ε-approximation of the efficient set is desired. In our computational experience, we shall apply phase two only to triangles where (5.3) does not hold, referred to as *large triangles*.

Assume that W^+ and W^- define a large triangle. First, the value of the parameter λ is found such that W^+ and W^- have the same parametric weight, i.e.

$$\lambda = \frac{W_2^+ - W_2^-}{W_1^- - W_1^+} \tag{5.4}$$

Next, we search the triangle using a K best strategies procedure on $\mathcal{H}_\lambda$ (e.g. procedure *K-BSreopt* in Figure 4.8). For the k'th strategy, we find its corresponding weights W_1 and W_2 using criterion one and two and use procedure $add(\Phi, W)$ to update Φ. The procedure stops when the current upper bound is reached. Recall that the upper bound can be decreased during the search, if new nondominated points are found. Moreover, if only an ε-approximation of $\mathcal{W}_{\mathit{eff}}$ is needed, i.e. procedure $phaseOne(\varepsilon)$ has been used in the first phase, then the upper bound can be decreased, since, instead of considering for instance the upper bound $ub_0 = W_1^-\lambda + W_2^+$, we can now consider the upper bound $ub_0 = (1-\varepsilon)\, W_1^-\lambda + (1-\varepsilon)\, W_2^+$ (see Figure 5.5).

Clearly, the efficiency of phase two depends on how many points of $\mathcal{W}$ lie inside the triangle. Unfortunately, under expectation criteria, the increase in the weight when using the K best strategies procedure is very small as pointed out in Section 4.3, and hence searching a triangle may be unacceptably slow. In order to overcome this difficulty, it is necessary to reduce the number of hyperpaths generated by the K best strategies procedure, i.e. we are interested in finding ways to prune the candidate set of the K best strategies procedure.

Consider Branching Operation 4.1.2 on the k'th hyperpath π^k in hypergraph $\mathcal{H}^k$ with valid ordering $V_{\pi^k} = (s, u_1, ..., u_{q_k})$ and subhypergraphs $\mathcal{H}^{k,i}$, $i = 1, ..., q_k$, defined in Definition 4.1.2. Let $m_j^{k,i}(u)$ denote the weight of the minimum weight s-u hyperpath in $\mathcal{H}^{k,i}$ when using criterion j.

Our goal is now to detect situations where the hyperpaths in $\mathcal{H}^{k,i}$ can be discarded from the candidate set. It is obvious that all hyperpaths in subhypergraph $\mathcal{H}^{k,i}$ must correspond to a point (W_1, W_2) greater than or equal to $(m_1^{k,i}(t), m_2^{k,i}(t))$. Hence if

$$m_1^{k,i}(t) \geq W_1(\pi^k) \textbf{ and } m_2^{k,i}(t) \geq W_2(\pi^k)$$

then all hyperpaths in $\mathcal{H}^{k,i}$ are dominated by π^k and can be discarded. Here we consider the hyperpath π^k on which the branching operation is performed. A stronger rule can be obtained by considering the whole set Φ of nondominated points, currently found in the first and second phase. Moreover, we may adopt ε-dominance, rather than pure dominance.

Rule 5.4.1 *Assume* $\exists \hat{W} \in \Phi$ *satisfying* $m_1^{k,i}(t) \geq (1-\varepsilon)\hat{W}_1$ ***and*** $m_2^{k,i}(t) \geq (1-\varepsilon)\hat{W}_2$. *Then for all hyperpaths* π *in* $\mathcal{H}^{k,i}$, *we have that* $\hat{W}$ ε*-dominates* $W(\pi)$.

Proof Since all hyperpaths π in $\mathcal{H}^{k,i}$ must correspond to a point satisfying

$$(W_1(\pi), W_2(\pi)) \geq (m_1^{k,i}(t), m_2^{k,i}(t))$$

we have that, if $\hat{W}$ ε-dominates $(m_1^{k,i}(t), m_2^{k,i}(t))$, then it also ε-dominates $W(\pi)$. ■

While searching a triangle defined by W^+ and W^- only points inside the triangle are of interest. The following rule also considers ε-dominance.

Rule 5.4.2 *If $m_1^{k,i}(t) \geq W_1^-(1-\varepsilon)$ **or** $m_2^{k,i}(t) \geq W_2^+(1-\varepsilon)$, then all hyperpaths in $\mathcal{H}^{k,i}$ correspond to points either outside the triangle or ε-dominated by W^+ or W^-.*

Proof Assume that $m_1^{k,i}(t) \geq W_1^-(1-\varepsilon)$. If $m_1^{k,i}(t) < W_1^-$ and $m_2^{k,i}(t) < W_2^+$, then W^- ε-dominates $(m_1^{k,i}(t), m_2^{k,i}(t))$ and hence also $W(\pi)$, for all hyperpaths π in $\mathcal{H}^{k,i}$; otherwise if $m_1^{k,i}(t) \geq W_1^-$, then $(m_1^{k,i}(t), m_2^{k,i}(t))$ is outside the triangle and hence will $W(\pi)$, for all hyperpaths π in $\mathcal{H}^{k,i}$ also be outside the triangle. The proof is similar if we assume that $m_2(t) \geq W_2^+(1-\varepsilon)$. ■

Note that, if we remove the subhypergraphs satisfying Rule 5.4.1 and 5.4.2 from the candidate set in the K best strategies procedure, we only remove hyperpaths corresponding to either points which are ε-dominated by a point in Φ or points outside the triangle. Therefore the set of nondominated points found in the triangle when the K best strategies procedure stops, is an ε-approximation of the true set of nondominated points in the triangle. This gives us the following theorem.

Theorem 5.4.4 *Given $\varepsilon \geq 0$, the set of nondominated points Φ found by phase one using procedure phaseOne(ε) and phase two using Rule 5.4.1 and 5.4.2 is an ε-approximation of $\mathcal{W}_{\text{eff}}$.*

Unfortunately, in most cases Rule 5.4.1 and 5.4.2 do not remove enough hyperpaths from the candidate set to speed up the triangle search significantly. Therefore we must adopt an *approximated* triangle search procedure. Given ε, the goal here is not to find an ε-approximation of $\mathcal{W}_{\mathit{eff}}$; instead we simply want to prevent the K best strategies procedure from getting stuck due to the huge number of almost equivalent hyperpaths. The basic idea is quite simple: when branching on hyperpath π^k, we do not want to consider hyperpaths corresponding to points that are "too close" to $W(\pi^k)$.

For each criterion $j \in \{1,2\}$, let $W_j^k(u)$, $u \in \mathcal{V}_{\pi^k}$, denote the weights of the nodes in π^k. Recall that efficient hyperpaths may not necessarily contain efficient subhyperpaths. However, removing subhypergraphs $\mathcal{H}^{k,i}$ where all s-t hyperpaths contain inefficient s-u_i hyperpaths may be useful. The following rule considers ε-dominance in node u_i

Rule 5.4.3 *If $m_1^{k,i}(u_i) \geq (1-\varepsilon) W_1^k(u_i)$ and $m_2^{k,i}(u_i) \geq (1-\varepsilon) W_2^k(u_i)$ then all s-u_i subhyperpaths in $\mathcal{H}^{k,i}$ are ε-dominated by $W^k(u_i)$ and we discard subhypergraph $\mathcal{H}^{k,i}$.*

Let $W_j^{k,i}(u)$ denote the weights of a minimum parametric weight hyperpath $\pi^{k,i}$ in $\mathcal{H}^{k,i}$ when considering criterion j. According to Theorem 4.1.6, we have that the weight of hyperpath $\pi^{k,i}$ using criterion j is

$$W_j^{k,i}(t) = W_j^k(t) + \left(W_j^{k,i}(u_i) - W_j^k(u_i)\right) f^\eta(u_i)$$

where $f^\eta(u_i)$ is found using (2.10) on the end-tree $\eta^{k,i}$ defined in Definition 4.1.1. Hence an improvement in criterion j is obtained if $(W_j^{k,i}(u_i) - W_j^k(u_i)) < 0$. However, the improvement may be small if $f^\eta(u_i)$ is small. This may often be the case as pointed out in Section 4.3. Therefore one way to prevent finding points close to $W(\pi^k)$ is to skip subhypergraph $\mathcal{H}^{k,i}$ when $f^\eta(u_i)$ is too small. Let us denote by ε_1 a lower bound on $f^\pi(u_i)$. We consequently have the following very simple rule.

Rule 5.4.4 *If $f^{\eta}(u_i) \leq \varepsilon_1$ then discard subhypergraph $\mathcal{H}^{k,i}$.*

Consider a node v in $\mathcal{H}^{k,i}$ corresponding to node u in G at time t. Recall that $f^{\eta}(v)$ is the probability of arriving at node u at time t when following the strategy defined by $\pi^{k,i}$ according to Theorem 3.4.1. That is, the s-v hyperpath contained in $\pi^{k,i}$ defines a *sub-strategy* for travelling from u to the destination, leaving at time t; this sub-strategy has a probability $f^{\eta}(v)$ of being used. Our hypergraph model cannot discriminate between sub-strategies that occur with low or high probability. However, using Rule 5.4.4, low probability sub-strategies are not examined. This approach may be quite reasonable in an online setting, where a situation such as "leave node u at time t" would be considered only when - and if - the situation occurs.

Even if Rule 5.4.4 fails, we may skip subhypergraph $\mathcal{H}^{k,i}$ if, for both criteria, the actual improvement in the weights of the minimum parametric weight hyperpath in $\mathcal{H}^{k,i}$ is small. The maximal improvement for criterion j at node u_i is $W_j^k(u_i) - m_j^{k,i}(u_i)$. As discussed above, this gives an improvement $(W_j^k(u_i) - m_j^{k,i}(u_i))f^{\eta}(u_i)$, at node t. If this improvement is small for both criteria, we may skip subhypergraph $\mathcal{H}^{k,i}$. In the following rule, the improved weights at t for both criteria are compared to some previously found nondominated point. Here, ε_2 denotes a lower bound on the improvement.

Rule 5.4.5 *Assume $\exists \hat{W} \in \Phi$ satisfying $W_1\left(\pi^k\right) - (W_1^k(u_i) - m_1^{k,i}(u_i))f^{\eta}(u_i) \geq (1-\varepsilon_2)\hat{W}_1$ **and** $W_2\left(\pi^k\right) - (W_2^k(u_i) - m_2^{k,i}(u_i))f^{\eta}(u_i) \geq (1-\varepsilon_2)\hat{W}_2$. Then discard subhypergraph $\mathcal{H}^{k,i}$.*

Rule 5.4.5 is very restrictive, since, by skipping subhypergraph $\mathcal{H}^{k,i}$, we assume that no efficient hyperpath in $\mathcal{H}^{k,i}$ can be found by changing the predecessor $p(v)$ of a node $v \in E_{\eta} \setminus \{u_i\}$. This may often be the case and hence using Rule 5.4.5 may result in poor approximations. Therefore Rule 5.4.5 should only be used when the K best strategies procedure begins to stall.

Clearly, the set of points determined by Rules 5.4.3-5.4.5 ε-dominates the true set of nondominated points for some ε. But we cannot determine how good the approximation is, i.e. the value of ε. However, an upper bound on ε can be found for each triangle examined.

Theorem 5.4.5 *Consider an approximation $\Phi = \left\{W^1, ..., W^q\right\}$ of a large triangle where W^1 and W^q denote the extreme nondominated points defining the triangle. Then Φ is an ε_{ub}-approximation of the true set of nondominated points in the triangle with*

$$\varepsilon_{ub} = \max_{i=1,...,q-1} \left\{ 1 - \frac{\gamma\left(W^1, \lambda\right)}{W_1^{i+1}\lambda + W_2^i} \right\} \tag{5.5}$$

where λ is the value such that W^1 and W^q have the same parametric weight.

Proof Consider the triangle in Figure 5.8 with nondominated points $\Phi = \left\{W^1, ..., W^q\right\}$ ($q = 4$). Recall that W^1 and W^q are extreme nondominated points. Therefore is suffices to find the minimum ε needed such that a point on the segment joining W^1 and W^q is always dominated by a point in Φ. That is, according to Definition 5.1.6 we have to find an ε_{ub} so that the points $(1-\varepsilon_{ub})W^1, ..., (1-\varepsilon_{ub})W^q$, marked with crosses in Figure 5.8, dominate all the points in the segment joining W^1 to W^q. This is the case if the points $\left((1-\varepsilon_{ub})W_1^{i+1}, (1-\varepsilon_{ub})W_2^i\right)$, $i = 1, ..., q-1$, marked with squares in Figure 5.8 have

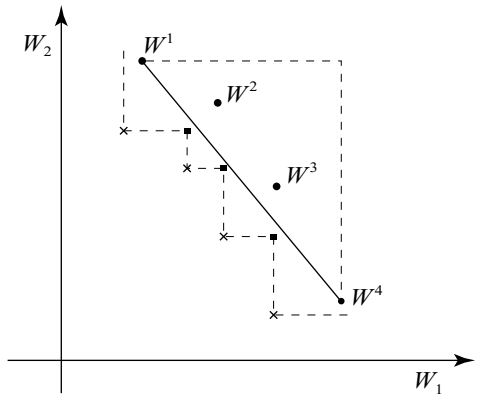


Figure 5.8: Finding an upper bound on ε for a triangle.

parametric weight below the parametric weight $\gamma\left(W^1,\lambda\right)$ of the segment joining W^1 to W^q, i.e.

$$\left(1-\varepsilon_{ub}\right)W_1^{i+1}\lambda+\left(1-\varepsilon_{ub}\right)W_2^i\leq\gamma\left(W^1,\lambda\right),\quad i=1,...,q-1$$

Modifying the inequality and taking the maximum value of ε_{ub}, we get (5.5). ■

Finally, a few remarks about the data structures needed in phase two. An ordered nondominated set Φ must be maintained. For each triangle, we run a modified version of procedure *K-BSreopt* ranking hyperpaths according to the parametric weight. For each subhypergraph $\mathcal{H}^{k,i}$ added to the candidate, we store, for $j=1,2$, weights $m_j^{k,i}(t)$ and $W_j(\pi^{k,i})$ and the parametric weight $\gamma(W(\pi^{k,i}),\lambda)$. Assume that the k'th hyperpath π^k is picked from the candidate set. Using Theorem 4.1.6, we now can find $m_j^{k,i}(t)$ and $W_j^{k,i}(t)$ and check Rules 5.4.1-5.4.5 (if they are used).

5.4.2 Min-max criteria

Assume that both criteria are min-max criteria, i.e. we have to find efficient hyperpaths using two distance weighting functions. Unfortunately Theorem 5.4.1 cannot be extended to hypergraphs when two distance weighting functions are considered, as can be seen in the following example

Example 5 (continued) Consider the time-expanded hypergraph with costs as shown in Figure 5.6 on page 86 and assume that both criteria is MMC. A minimum weight hyperpath π of $\mathcal{H}_\lambda$ ($\lambda=1$), using the distance weighting function, is shown in Figure 5.9. The costs $(w_1(e),w_2(e),w_\lambda(e))$ are shown near each hyperarc e. The minimum weight of π, using weights $w_\lambda(e)$, is $W_\lambda(\pi)=14$. However, the weight of π using criterion one and two is $W_1(\pi)=9$ and $W_2(\pi)=7$, i.e. the parametric weight $\gamma(W(\pi),\lambda)$ is equal to 16. ■

Note that, we have $W_\lambda(\pi)\leq\gamma(W(\pi),\lambda)$ in the above example. This property holds true in general.

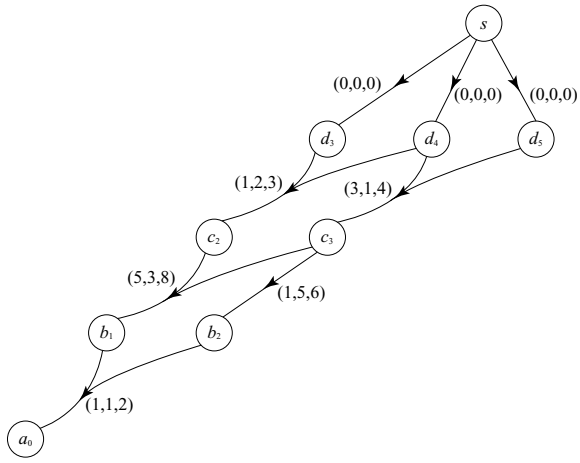


Figure 5.9: A minimum weight hyperpath of $\mathcal{H}_\lambda$.

Theorem 5.4.6 *Let $W_\lambda(\pi)$ denote the distance of a hyperpath π in $\mathcal{H}_\lambda$. For every $\lambda > 0$ and for every $\pi \in \Pi$, we have that $W_\lambda(\pi) \leq \gamma(W(\pi), \lambda)$.*

Proof For each u in π, denote by $W_\lambda(u)$, $W_1(u)$ and $W_2(u)$ the distance of node u in π with respect to the weights w_λ, w_1 and w_2, respectively. Consider a valid ordering $V_\pi = (u_1, ..., u_q)$ for π. We shall prove by induction that for each u_j in V, $W_\lambda(u_j) \leq W_1(u_j)\lambda + W_2(u_j)$. The property clearly holds for $u_1 = s$. Now, assume that the property holds for each node preceding $u = u_j$ in V. Then

$$\begin{aligned} W_\lambda(u) &= \max_{v \in T(p(u))} \{W_\lambda(v)\} + w_\lambda(p(u)) \\ &= \max_{v \in T(p(u))} \{W_\lambda(v)\} + w_1(p(u))\lambda + w_2(p(u)) \\ &\leq \big(\max_{v \in T(p(u))} \{W_1(v)\} + w_1(p(u))\big)\lambda \; + \; \big(\max_{v \in T(p(u))} \{W_2(v)\} + w_2(p(u))\big) \\ &= W_1(u)\lambda + W_2(u). \end{aligned}$$

Hence we have that $W_\lambda(\pi) \leq \gamma(W(\pi), \lambda)$. ∎

A better lower bound can be found, if all hyperpaths in $\mathcal{H}_\lambda$ must contain an end-tree. Note that this is the case when considering the subhypergraphs $\mathcal{H}^{k,i}$ created when using Branching Operation 4.1.2 on the k'th minimum weight hyperpath π^k. All hyperpaths in $\mathcal{H}^{k,i}$ must contain end-tree $\eta^{k,i}$ defined in Definition 4.1.1.

Consider subhypergraph $\mathcal{H}^{k,i}$ and denote by $W_\lambda^{k,i}(v)$, $W_1^{k,i}(v)$ and $W_2^{k,i}(v)$ the weight of node v in π with respect to the weights $w_\lambda(e)$, $w_1(e)$ and $w_2(e)$, respectively. Similarly, let $l_\lambda^\eta(v)$, $l_1^\eta(v)$ and $l_2^\eta(v)$ denote the maximal weight of a v-t path in $\eta^{k,i}$ with respect to the weights $w_\lambda(e)$, $w_1(e)$ and $w_2(e)$, respectively

Theorem 5.4.7 *Given hyperpath π in $\mathcal{H}^{k,i}$ let*

$$\hat{W}_\lambda(\pi) = \max_{v \in E_\eta} \{W_\lambda(v) + l_1^\eta(v)\lambda + l_2^\eta(v)\} \tag{5.6}$$

For every $\lambda > 0$ *and for every hyperpath* π *in* $\mathcal{H}^{k,i}$ *we have that* $W_\lambda(\pi) \leq \hat{W}_\lambda(\pi) \leq \gamma(W(\pi), \lambda)$.

Proof Consider the valid ordering $V = (u_{i+1}, ..., u_q = t)$ of $\eta^{k,i}$. We shall first prove by induction that, for each u_j in V, $l^\eta_\lambda(u_j) \leq l^\eta_1(u_j)\lambda + l^\eta_2(u_j)$. The property clearly holds for $u_q = t$. Now assume that the property holds for each node succeeding u in V. Then

$$\begin{aligned} l^\eta_\lambda(u) &= \max_{e \in FS_\eta(u)} \{l^\eta_\lambda(h(e)) + w_\lambda(e)\} \\ &= \max_{e \in FS_\eta(u)} \{l^\eta_\lambda(h(e)) + w_1(e)\lambda + w_2(e)\} \\ &\leq \max_{e \in FS_\eta(u)} \{l^\eta_1(h(e)) + w_1(e)\}\lambda + \max_{e \in FS_\eta(u)} \{l^\eta_2(h(e)) + w_2(e)\} \\ &= l^\eta_1(u)\lambda + l^\eta_2(u) \end{aligned}$$

According to Theorem 2.4.3, the weight $W_\lambda(\pi)$ is

$$\begin{aligned} W_\lambda(\pi) &= \max_{v \in E_\eta} \{W_\lambda(v) + l^\eta_\lambda(v)\} \\ &\leq \max_{v \in E_\eta} \{W_\lambda(v) + l^\eta_1(v)\lambda + l^\eta_2(v)\} \\ &\leq \max_{v \in E_\eta} \{W_1(v) + l^\eta_1(v)\}\lambda + \max_{v \in E_\eta} \{W_2(v) + l^\eta_2(v)\} \\ &= \gamma(W(\pi), \lambda) \end{aligned}$$

Hence we have that $W_\lambda(\pi) \leq \hat{W}_\lambda(\pi) \leq \gamma(W(\pi), \lambda)$. ■

Note that, if only one v-t path exists in $\eta^{k,i}$ for each $v \in E_\eta$, then $l^\eta_\lambda(u) = l^\eta_1(u)\lambda + l^\eta_2(u)$ and hence $W_\lambda(\pi) = \hat{W}_\lambda(\pi)$ and we obtain no improvement using $\hat{W}_\lambda(\pi)$.

Since $\hat{W}_\pi(\pi(\lambda))$ is a lower bound on $\gamma(\lambda)$, Theorem 5.4.7 provides us with the following corollary

Corollary 5.4.2 *Let* π *denote a minimum weight hyperpath in* $\mathcal{H}_\lambda$ *using* (5.6). *Then* $\hat{W}_\lambda(\pi) \leq \gamma(\lambda)$.

That is, using Corollary 5.4.2 results in a minimum weight hyperpath π with weight $\hat{W}_\lambda(\pi)$ which is a lower bound on $\gamma(\lambda)$. However, we cannot find the minimum parametric weight hyperpath $\pi(\lambda)$ by just solving a minimum weight hyperpath problem. Instead π only provides an approximation of the supported nondominated point corresponding to $\pi(\lambda)$.

As we will see, computational results show that the lower bound $\hat{W}_\lambda(\pi)$ often provides us with a poor approximation. Therefore a greedy heuristic procedure for finding a hyperpath providing an approximation of $\pi(\lambda)$ was developed. The procedure *SHTgreedy*, shown in Figure 5.10, is similar to procedure *SHTacyclic*. However, here we maintain 3 labels for each node: the weight $W_j(v)$, $j = 1, 2$, of the current hyperpath (defined by p) to node v using criterion j and the parametric weight $W(v)$ of the current hyperpath to node v. When considering node v in the procedure, the predecessor hyperarc is set to the hyperarc e yielding the minimal parametric weight

$$(F_1(e) + w_1(e))\lambda + F_2(e) + w_2(e)$$

```
1  procedure SHTgreedy(s, λ, H)
2      W1(v1) := W2(v1) := W(v1) := 0;
3      for (i = 2 to n) do W1(v1) := W2(v1) := W(v1) := ∞;
4      for (i = 2 to n) do
5          for (e ∈ BS(vi)) do
6              if (W(vi) > wλ(e) + F1(e)λ + F2(e)) then
7                  W(vi) := wλ(e) + F1(e)λ + F2(e); p(vi) := e;
8                  W1(vi) := w1(e) + F1(e); W2(vi) := w2(e) + F2(e);
9              end if
10         end for
11     end for
12 end procedure
```

Figure 5.10: A greedy heuristic for finding an approximation of $\pi(\lambda)$.

where $F_j(e)$ denotes the function (2.2) using weights $W_j(u)$, $j = 1, 2$. Clearly, procedure *SHTgreedy* does not find $\pi(\lambda)$ since it assumes that all nodes in a minimum parametric weight hyperpath have minimum parametric weight, too. Nevertheless, it often provides a solution quite close to $\pi(\lambda)$, as we will see in Section 5.6. Note that the weight $W(t)$, when procedure *SHTgreedy* stops, is equal to the parametric weight $\gamma(W(\pi), \lambda)$ of the s-t hyperpath π defined by p. Hence $W(t)$ is an upper bound on $\gamma(\lambda)$. In the following we therefore let W_λ^{ub} denote the weight of the hyperpath π found by procedure *SHTgreedy*.

Phase one: Finding an approximated frontier

Due to the fact that we cannot find the minimum parametric weight hyperpath by solving a minimum weight hyperpath problem on $\mathcal{H}_\lambda$, we cannot find the frontier using procedure *phaseOne* in Figure 5.4. However, an approximated frontier can be found for a given λ by finding the weight W_λ of the minimum hyperpath of $\mathcal{H}_\lambda$ and the weight W_λ^{ub} of the hyperpath returned by procedure *SHTgreedy*. That is, procedure *phaseOne* is modified so that, for each new search direction λ, we find W_λ and W_λ^{ub} and add them to Φ using procedure $add(W, \Phi)$. The procedure stops when no new points can be found. Note that Φ may contain only a subset of the extreme nondominated points. Furthermore, it may also contain points dominated by points in $\mathcal{W}_{\mathit{eff}}$. Nevertheless, the approximate frontier Φ defines a set of triangles which can be used in phase two. Moreover, if an interactive approach is used, Φ can be used to guide the decision maker.

Phase two: Looking into the approximated triangles

After phase one an ordered nondominated set Φ, defining a set of triangles we search for further nondominated points in phase two, has been found.

Assume that we consider the triangle defined by W^+ and W^- in Φ and λ given in (5.4). Unfortunately we cannot rank hyperpaths in non-decreasing order of the parametric weight by using a K best strategies procedure on $\mathcal{H}_\lambda$. Due to Corollary 5.4.2, however, ranking hyperpaths according to $\hat{W}_\lambda(\pi)$ corresponds to ranking hyperpaths according to a lower bound on the parametric weight. Hence the following holds

Theorem 5.4.8 *Assume that we find the K minimum weight hyperpaths ranking hyperpaths according to the lower bound weight* $\hat{W}_\lambda(\pi)$. *Moreover, assume that the k'th hyperpath is selected with weight* $\hat{W}_\lambda(\pi^k)$ *then all hyperpaths* $\pi \in \Pi$ *with parametric weight* $\gamma(W(\pi), \lambda)$ *below* $\hat{W}_\lambda(\pi^k)$ *have been considered.*

Due to Theorem 5.4.8, ranking hyperpaths according to $\hat{W}_\lambda$ until the upper bound of the triangle is reached, provides us with a complete method, that is, all the hyperpaths with parametric weight below the upper bound would be obtained. Clearly, if $\hat{W}_\lambda(\pi)$ is a week lower bound on the parametric weight $\gamma(W(\pi), \lambda)$ then hyperpaths π with $\hat{W}_\lambda(\pi)$ below the upper bound of the triangle may have parametric weight $\gamma(W(\pi), \lambda)$ above the upper bound. As a result the K minimum weight hyperpaths procedure may have to consider a lot of points above the triangle before it stops. Computational results in Section 5.6 show that this may often be the case.

Another possibility is to find hyperpaths using procedure *SHTgreedy* within procedure *K-BSreopt*. That is, for each subhypergraph $\mathcal{H}^{k,i}$, we insert the hyperpath π found by procedure *SHTgreedy* into the candidate set. As discussed earlier, hyperpath π provides us with an approximation on $\pi(\lambda)$ in $\mathcal{H}^{k,i}$ with upper bound weight $W_\lambda^{ub}(\pi)$. Hence by using this approach, we find an approximation of the nondominated points in the triangle. Note that, by using procedure *SHTgreedy* within procedure *K-BSreopt* we clearly do not rank strategies exactly. That is, the weight of the k'th hyperpath found by procedure *K-BSreopt* may be greater than the weight of the $k+1$'th hyperpath[†].

It should also be pointed out that new extreme nondominated points may be found during phase two, since we only find an approximation of the frontier in phase one. If this happens the current set of extreme nondominated points is updated and the current K best strategies procedure is stopped and restarted on the triangles defined by the new point. Other choices would be possible, but computational testing shows that this choice is acceptable in terms of computation time.

As under expectation criteria, the efficiency of phase two depends on how many points of $\mathcal{W}$ lie inside the triangle. Under min-max criteria, the increase in the weight when using the K minimal hyperpath procedure is slow, since there may exist a lot of hyperpaths corresponding to the same point in the criterion space as pointed out in Section 4.3. Therefore rules have to be used to prune the candidate set of hyperpaths. The rules used under expectation criteria can also be used under min-max criteria except Rule 5.4.4. Note that, if hyperpaths are ranked according to lower bound weight $\hat{W}_\lambda(\pi)$ when triangles are searched and only Rule 5.4.1 and 5.4.2 is used, then Theorem 5.4.4 still holds.

A specialized branching operation for min-max criteria

In this section we present a new branching operation for min-max criteria. Branching Operation 4.1.2 has the following weakness when min-max criteria are considered.

Consider Branching Operation 4.1.2 on hyperpath π^k and let P_j denote a path contained in π^k with maximal weight $W_j(\pi^k)$ using criterion j. Recall that $\mathcal{H}^{k,i}$ with minimal hyperpath $\pi^{k,i}$ is obtained by removing the predecessor hyperarc $p^k(u_i)$ from node u_i (see Definition 4.1.2) Assume that $u_i \notin P_1 \cup P_2$. Then, according to Corollary 2.4.2 the point $W(\pi^{k,i})$ satisfies

$$W_j\left(\pi^{k,i}\right) \geq W_j\left(\pi^k\right), \quad j = 1, 2$$

[†]See Figure 5.12 on page 111 where an example of the ranking of the parametric weight is shown.

That is, $\pi^{k,i}$ will either contain the same maximum weight paths P_1 and P_2 as in π^k or paths with a higher weight. This suggests the following improved branching operation

Branching Operation 5.4.1 *Given hyperpath $\pi \in \Pi$ defined by predecessor function p with valid ordering V_π, let $V_{P_1P_2} = (s, \bar{u}_1, \bar{u}_2, \ldots, \bar{u}_r = t) \subseteq V_\pi$ denote a valid ordering of the nodes in $P_1 \cup P_2$. Moreover, let $\Pi_{P_1P_2}$ denote the set of s-t hyperpaths in $\mathcal{H}$ containing P_1 and P_2. Then the set $\Pi \backslash \Pi_{P_1P_2}$ can be partitioned into r disjoint subsets $\bar{\Pi}^i$, $1 \leq i \leq r$ as follows*

1. *Hyperpaths in $\bar{\Pi}^r$ do not contain hyperarc $p(\bar{u}_r)$, that is $p(t)$.*
2. *For $1 \leq i < r$, hyperpaths in $\bar{\Pi}^i$ contain hyperarcs $p(\bar{u}_j)$, $i+1 \leq j \leq r$, and do not contain hyperarc $p(\bar{u}_i)$.*

Proof Similar to the proof of Branching Operation 4.1.2. ∎

By using Branching Operation 5.4.1, all hyperpaths containing P_1 and P_2 are removed in one branching operation, since they all correspond to points dominated by or equal to point $W(\pi)$. It is obvious that hyperpaths in $\bar{\Pi}^i$ must contain the following end-tree $\bar{\eta}^i = (\mathcal{V}_\eta, \mathcal{E}_\eta)$, $i = 1, ..., r$ with

$$\mathcal{E}_\eta = \begin{cases} \emptyset & i = r \\ \{p(\bar{u}_r), ..., p(\bar{u}_{i+1})\} & \text{otherwise} \end{cases}, \quad \mathcal{V}_\eta = \begin{cases} \{t\} & i = r \\ \bigcup_{e \in \mathcal{E}_\eta} T(e) \cup h(e) & \text{otherwise} \end{cases}$$

As in Branching Operation 4.1.2, finding a minimum weight s-t hyperpath $\bar{\pi}^i \in \bar{\Pi}^i$ reduces to solving a minimum weight hypertree problem on a subhypergraph $\bar{\mathcal{H}}^i$, containing end-tree $\bar{\eta}^i$ of $\mathcal{H}$, defined by

Definition 5.4.1 Given π, let subhypergraph $\bar{\mathcal{H}}^i$, $i = 1, ..., r$, be obtained from $\mathcal{H}$ as follows

1. For each node $\bar{u}_j$, $i+1 \leq j \leq r$, remove each hyperarc in $BS(\bar{u}_j)$ except $p(\bar{u}_j)$.
2. Remove hyperarc $p(\bar{u}_i)$ from $BS(\bar{u}_i)$.

Since each s-t hyperpath in $\bar{\mathcal{H}}^i$ must contain an end-tree $\bar{\eta}^i$ and $\bar{\eta}^{i+1} \subset \bar{\eta}^i$, the weight $l^\eta_\lambda(v)$ (and the lengths $l^\eta_1(v)$ and $l^\eta_2(v)$), can be updated recursively during the branching operation on π. However, note that node $\bar{u}_i \in E_\eta$ is not necessarily the last node in the valid sub-ordering $V_{E_\eta} \subset V_\pi$. Therefore Corollary 2.4.2 cannot be used to find the weight of the minimal hyperpath $\bar{\pi}^i$ in $\bar{\mathcal{H}}^i$. This is due to the fact that the weight of the nodes $v \in E_\eta$ (above $\bar{u}_i$ in V_{E_η}) may change when the weight in node $\bar{u}_i$ is changed. Instead, we may do the following to find the updated weights of the minimal hyperpath $\bar{\pi}^i$ in $\bar{\mathcal{H}}^i$

1. First see if Theorem 2.4.4 can be used.
2. Otherwise update the weights for the nodes $v \in E_\eta$ (above $\bar{u}_i$ in the valid ordering of E_η) by using a modified version of procedure *SHTacyclic*.
3. Use Theorem 2.4.3 to calculate the weight in node t.

Unfortunately, item 2 above seems to be too time consuming. Instead we will follow a different approach. Consider the subhypergraph $\bar{\mathcal{H}}^i$ and assume that we add hyperpaths to the candidate set using the lower bound $\hat{W}_\lambda$ given in (5.6). Now, consider the weight $W_\lambda(\bar{u}_i)$. Since we remove a hyperarc in the backward star of node $\bar{u}_i$, the updated weight in node $\bar{u}_i$ will increase. As a result, the updated weight in the nodes $v \in E_\eta$ cannot decrease. Therefore using Corollary 2.4.2 with the updated weight in node $\bar{u}_i$ provides us with a lower bound on the weight $\hat{W}(\bar{\pi}^i)$ in $\bar{\mathcal{H}}^i$. We can now modify procedure *K-BSreopt* in Figure 4.8 so that Branching Operation 5.4.1 can be used.

1. Pick the minimal branching tree node τ (line 8) and create the subhypergraph corresponding to τ (line 10) as before.

2. Line 11 needs to be modified to

   (a) First, recalculate the weight W_τ of the hyperpath π_τ using procedure *SHTacyclic*.

   (b) If the weight W_τ is above the lowest weight in the candidate set then reinsert the subhypergraph into the candidate set with weight W_τ updated and start a new iteration.

   (c) Otherwise, output the hyperpath.

Similar item 1 and 2 above can be used when ranking hyperpaths using the upper bound W_λ^{ub} found by procedure *SHTgreedy* where item 2a uses procedure *SHTgreedy* instead of procedure *SHTacyclic*.

Note that the way we calculate weights and reinsert subhypergraphs is similar to procedure *K-BPSreopt* under a priori route choice. Moreover, when using *K-BSreopt* with Branching Operation 5.4.1, the rules in Section 5.4.1 cannot be applied as before, because we only calculate a lower bound on the minimal hyperpath for each subhypergraph when branching on hyperpath π_τ. Hence only Rule 5.4.3 can be applied to the nodes $\bar{u}_i \in P_1 \cup P_2$ when branching on hyperpath π_τ while Rules 5.4.1, 5.4.2 and 5.4.5 must be applied to the minimal hyperpath in π_τ found when considering the branching tree node τ. That is, the rules are mainly applied to hyperpaths when picked from the candidate set and not to the hyperpaths of the subhypergraphs created when branching.

5.4.3 Expectation criterion and min-max criterion

Assume that one of the criteria is expectation and the other criterion is min-max, i.e. we have to find efficient hyperpaths using a mean weighting function for the expectation criterion and a distance weighting function for the min-max criterion.

As in the case where min-max criteria is used, Theorem 5.4.1 cannot be extended to hypergraphs if one of the criteria is expectation and the other criterion is min-max. However, a lower bound on the parametric weight can be found.

Theorem 5.4.9 *Let $W_\lambda(\pi)$ denote the weight of a hyperpath π in $\mathcal{H}_\lambda$ when using the mean weighting function. For every $\lambda > 0$ and for every $\pi \in \Pi$, we have that $W_\lambda(\pi) \leq \gamma(W(\pi), \lambda)$.*

Proof The proof is not given, but it follows the same kind of reasoning as the proof of Theorem 5.4.6. ■

Due to Theorem 5.4.9, ranking hyperpaths according to weight W_λ in (5.6) until the upper bound of the triangle is reached provides us with a complete method. That is, all the efficient hyperpaths with parametric weight below the upper bound would be obtained. Moreover, the weight of the hyperpath returned by procedure *SHTgreedy*, shown in Figure 5.10, still provides us with an upper bound on the parametric weight. Hence adding hyperpaths to the candidate set using procedure *SHTgreedy* provides us with an approximation of the nondominated points in the triangle.

Using the above lower and upper bounds we can apply phase one and two as under min-max criteria.

5.5 Finding the set of efficient strategies under a priori route choice

In this section the problem of finding efficient strategies under a priori route choice is considered. The problem consists in finding the set of efficient path-strategies between an origin and a destination node when leaving the origin at time zero. The criteria given in Section 3.2 are used, i.e. we consider expectation criteria where the goal is to minimize the expected travel time (MET) or cost (MEC) or min-max criteria where the goal is to minimize the maximum possible travel time (MMT) or cost (MMC). We consider the case where waiting is allowed and the case where no waiting is allowed separately.

Since a path-strategy between an origin and a destination node corresponds to a hyperpath in the time-expanded hypergraph $\mathcal{H}$, finding the set of efficient path-strategies corresponds to finding the set of efficient hyperpaths in the time-expanded hypergraph $\mathcal{H}$, all corresponding to a path-strategy.

Let $\Pi_{PS} \subseteq \Pi$ denote the set of s-t hyperpaths corresponding to a path-strategy. Note that, we are not interested in finding the set of efficient hyperpaths Π_{eff} but the set of efficient hyperpaths in Π_{PS}. That is, we only consider a subset of Π, namely the hyperpaths corresponding to a path-strategy.

5.5.1 No waiting allowed

Consider an STD network where no waiting is allowed in the nodes in G and the problem of finding efficient hyperpaths corresponding to a path-strategy. Due to Corollary 4.2.3, the total number of path-strategies is significantly lower than the total number of strategies. As a result, we may expect that the K best path-strategy procedure searching a triangle has to find fewer path-strategies before reaching the upper bound of the triangle.

Assume that both criteria are expectation criteria, i.e. the resulting problem becomes finding efficient hyperpaths corresponding to a path-strategy using two mean weighting functions. Since the set Π_{PS} is a subset of Π, we have that Theorem 5.4.2 still holds. That is, we have the following corollary

Corollary 5.5.1 *Assume that both criteria are expectation criteria. Then finding the hyperpath $\pi \in \Pi_{PS}$ with minimum parametric weight, reduces to finding a minimum weight hyperpath $\pi \in \Pi_{PS}$ in $\mathcal{H}_\lambda$.*

Note that the minimum parametric weight hyperpath $\pi \in \Pi_{PS}$ defines a path-strategy corresponding to a supported nondominated point. Hence the frontier can be found in phase one by a modified version of procedure *phaseOne* in Figure 5.4; simply change procedure *SHTacyclic* to e.g. procedure *K-BPS_MB* with $K = 1$. Note that the first path-strategy, when using procedure *K-BPS_MB*, can be found relatively fast for "realistic" STD networks (see Section 4.3).

In phase two the only change, compared to time-adaptive route choice, is that here we just have to search each triangle using a K best path-strategies procedure instead of a K best strategies procedure. Rules 5.4.1 and 5.4.2 can still be used to remove subgraphs from the candidate set. This is not the case for Rules 5.4.3-5.4.5. In the experimental tests, however, no rules are used, since this is unnecessary due to the fact that the number of hyperpaths in Π_{PS} is significantly lower that the number of hyperpaths in Π.

Now assume that both criteria are min-max criteria, i.e. we have to find efficient hyperpaths $\pi \in \Pi_{PS}$ using two distance weighting functions. As under time-adaptive route choice, we may use a K best path-strategy procedure adding hyperpaths $\pi \in \Pi_{PS}$ to the candidate set by either using the lower bound weight $\hat{W}_\lambda$ in (5.6) or the upper bound weight W^{ub}_λ found by using procedure *SHTgreedy*. Due to Theorem 5.4.8 ranking hyperpaths according to $\hat{W}_\lambda$ until the upper bound of the triangle is reached, provides us with a complete method, that is, all the hyperpaths $\pi \in \Pi_{PS}$ with parametric weight below the upper bound would be examined. If, instead, W^{ub}_λ is used, we find an approximation of the nondominated points in the triangle.

Finally, if one of the criteria is expectation and the other criterion is min-max, we may again use either a lower bound weight which provides us with a complete method or an upper bound weight resulting in an approximation.

5.5.2 Waiting allowed

Consider an STD network where waiting is allowed in the nodes in G. If both criteria are expectation criteria, then Corollary 5.5.1 still holds and the frontier can be found in phase one as in the case where no waiting is allowed. For min-max criteria phase one can also be used as in the case where no waiting is allowed.

Recall that, if waiting is allowed, then a path P in G defines a possible exponential number of path-strategies, distinguished from each other by the use of waiting (see Section 4.2.2). Therefore there may exist an exponential number of hyperpaths $\pi \in \Pi_{PS}$ corresponding to P, each having different weights. That is, different nondominated points may correspond to the same path in G. As a result phase two has to search each triangle using a K best path-strategy procedure where Branching Operation 4.1.2 is used, if the branching operation branches on a hyperpath corresponding to a path-strategy. Unfortunately, the results in Section 4.3.3 show that the path-strategies found with the K best path-strategy procedure often correspond to the same path in G resulting in a small increase in the parametric weight. That is, the procedure searching a triangle begins to stall and we have to find a way to prune the candidate set.

A simple way of avoiding this is to consider one path-strategy only for each path in G, i.e. we do not use Branching Operation 4.1.2 in e.g. procedure *K-BPS_MB* (see Section 4.2.2). This approach provides us with an approximation of the nondominated set.

5.6 Computational results

In this section we report the computational experience with the procedures previously described in this chapter. The procedures have been implemented in C++ and tested on a 1 GHz PIII computer with 1GB RAM using a Linux Red Hat operating system. The programs have been compiled with the GNU C++ compiler with optimize option -O.

All tests are performed on time-expanded hypergraphs generated with the TEGP generator (see Section 3.5). In the following, we use the term *hypergraph class* as a particular setting of the TEGP input parameters except that different criteria and correlation between the criteria may be chosen for the same class. That is, for each hypergraph class, different instances of the problem can be generated by choosing different criteria, correlation and seeds.

For each hypergraph class, the following criteria are considered: MET/MEC, MEC/MEC and MMC/MMC (criterion one/criterion two). Three types of correlation between the two criteria are considered. If the first criterion considers time and the second cost (T/C), then the weight, with respect to the first criterion, is zero on all the hyperarcs and the second weight follows a pattern as shown in Figure 3.5 on page 30. If both criteria consider cost (C/C), then two different types of correlation between the costs are considered, namely no correlation (no_{cor}) and negative correlation (neg_{cor}) (see Section 3.5 for further details).

Peak dependent costs are used in all classes with an off-peak cost interval equal to $[lb_c, ub_c] = [1, 1000]$. The deviation mean ratio is set to $\rho = 0.25$ in all classes. The remaining input parameters will be specified when we consider the different classes used in the tests.

5.6.1 Performance measures/statistics

In this section performance measures/statistics used to evaluate the procedures are described. For each hypergraph class, type of criteria and correlation type, the measures are average or maximum over five independent runs using a different seed. The statistics can be divided into two groups. In group one the following statistics concerning phase one is reported. The abbreviation used in the tables is given in parentheses.

Frontier size ($|\Phi_f|$): The size of the frontier. Only reported when both criteria are expectation criteria.

Approximated frontier size ($|\Phi_f(\varepsilon)|$): The size of the approximated frontier using procedure *phaseOne*(ε) when expectation criteria are considered. If min-max criteria are considered, $|\Phi_f(\varepsilon)|$ is the number of extreme points in the nondominated set found in phase one.

CPU time (CPU_f): The CPU time used for phase one reported in seconds.

Number of triangles ($|\triangle|$): Number of large triangles searched by phase two when expectation criteria are considered and the number of triangles defined by the nondominated set when phase two stops when min-max criteria are considered.

Relative increase (RI_j): The relative increase from the upper/left point W^{ul} to the lower/right point W^{lr} for the j'th criteria defined as $(W_j^{lr} - W_j^{ul})/W_j^{ul}$. Reported in percent.

In the second group we report statistics for each triangle searched in phase two. The abbreviation used in the tables are given in parentheses.

CPU time ($CPU_{\triangle}$): The CPU time for searching a triangle reported in seconds. The average CPU time for all the triangles searched is reported. Moreover, also the maximum CPU time of all the triangles searched is reported.

Points in the triangle ($|\Phi_{\triangle}|$): The number of nondominated points in the triangle not including the two points defining the triangle. Average and maximum results reported.

Upper bound on epsilon (ε_{ub}): The upper bound (5.5) on epsilon for each triangle reported in percent. Only reported when expectation criteria are considered. Average and maximum results reported.

Number of unfinished triangles (U): The number of triangles where the search stopped because a limit on the number of iterations was reached, i.e. the K best strategies procedure terminated before reaching the upper bound defined by the triangle.

Number of new frontier points (new_f): Number of new frontier points found during phase two when min-max criteria are considered.

Epsilon ($\varepsilon_{\Phi s}$): The epsilon needed for the nondominated points found by the procedure to ε-dominate a triangle in a nondominated set Φ_s when min-max criteria are considered. Average and maximum results reported.

Number of iterations (ite): Average number of iterations performed when searching a triangle.

5.6.2 Time-adaptive route choice

In this section the problem of finding efficient strategies under time-adaptive route choice is considered. Our main goal here is to evaluate phase one and phase two under different criteria and different correlation between the two criteria.

The criteria given in Section 3.2 are used, i.e. we consider expectation criteria where the goal is to minimize the expected travel time (MET) or cost (MEC) or we consider min-max criteria where the goal is to minimize the maximum possible travel time (MMT) or cost (MMC).

We consider the case where both criteria are expectation criteria and the case where both criteria are min-max criteria separately. As we will see, using expectation criteria results in a dense set of nondominated points while using min-max criteria results in a sparse set of nondominated points. Moreover, the problem is easier to solve when using expectation criteria, i.e. when the nondominated set is dense. Therefore, one may expect that the case where one of the criteria is expectation and the other criterion is min-max, is easier to solve than the case, where two min-max criteria are used. This is confirmed by a few computational tests not presented in this thesis. Therefore the case where one of the criteria is expectation and the other criterion is min-max will not be considered.

Class	1	2	3	4	5	6	7	8
n	3342	2817	2314	1862	3342	3342	3342	3342
m_h	10976	9262	7612	6132	10976	10976	10976	10976
m_a	102	90	81	76	102	102	102	102
$\|T(e)\|$	4	5	6	6	4	4	4	4
H	144	144	144	144	144	144	144	144
I_T	[3,10]	[3,15]	[3,20]	[3,30]	[3,10]	[3,10]	[3,10]	[3,10]
I_C	[1,1000]	[1,1500]	[1,2000]	[1,2500]	[1,1050]	[1,1100]	[1,1200]	[1,1500]
ψ	0	0.5	1.0	1.5	0	0	0	0
r_ξ	0	0	0	0	0.05	0.10	0.20	0.50

Table 5.1: Hypergraph classes for preliminary tests (grid size 5×8).

Expectation criteria

Let both criteria be expectation criteria, i.e. we have to find efficient hyperpaths using two mean weighting functions.

First, some preliminary tests are carried out to point out the impact of some features of the TEGP generator, and to justify the relevant parameter settings. The effect of peak increase changes and changes in the range of the random perturbation are examined separately. An underlying grid size of 5×8 is used and no waiting is allowed. In all classes, a cycle consists of 144 time instances, i.e. 12 hours divided into 5 minute intervals. A cycle has two peaks, each with a total length of 5 hours with each part of the peak pk_j lasting 1 hour and 40 minutes. The first peak starts after half an hour ($t = 6$) and the interval of possible off-peak mean travel times is $[lb_t, ub_t] = [4, 8]$, i.e. an off-peak mean travel time between 20 and 40 minutes. A fixed time horizon of 144 time instances is used. Eight hypergraph classes are considered. In Table 4.1 the following statistics for class 1-8 are reported:

Number of nodes (n): Recall that the generator may generate nodes and hyperarcs in $\mathcal{H}$ which cannot be contained in an s-t hyperpath. These nodes and hyperarcs are removed from $\mathcal{H}$ in a preprocessing step. The number of nodes in $\mathcal{H}$ after preprocessing are reported.

Number of hyperarcs (m_h): Number of "true" hyperarcs in $\mathcal{H}$ after preprocessing (i.e. $|T(e)| > 1$).

Number of arcs (m_a): Number of arcs in $\mathcal{H}$ after preprocessing.

Tail size ($|T(e)|$): The average number of nodes in the tail of each hyperarc, i.e. the average number of elements in the travel time density of $X(u, v, t)$ when leaving node u at time t along arc (u, v).

Time horizon (H): The time horizon or number of time instances of the STD network.

Travel time interval (I_T): The interval of possible travel times (see (3.10)).

Cost interval (I_C): The interval of travel costs used (see (3.11)).

				$\lvert\Phi_\triangle\rvert$		$\text{CPU}_\triangle$		ε_{ub}	
Class	ψ	$\lvert\Phi_f\rvert$	$\lvert\triangle\rvert$	ave	max	ave	max	ave	max
T/C									
1	0	4	3	19	59	0.30	0.86	1.80	4.12
2	0.5	55	3	8	26	0.20	0.75	1.78	3.75
3	1.0	64	4	6	15	0.17	0.30	1.60	3.84
4	1.5	70	3	11	31	0.19	0.41	1.55	4.59
C/C (neg_{cor})									
1	0	7	6	135	690	11.42	122.08	1.55	10.46
2	0.5	140	14	20	306	4.13	168.89	1.23	10.59
3	1.0	138	9	78	589	17.49	189.48	1.44	4.84
4	1.5	154	8	70	308	14.42	182.26	1.88	15.42

Table 5.2: Preliminary tests – Changing the peak increase parameter ψ (expectation criteria).

The value of the peak increase parameter ψ and the range of the random perturbation r_ξ are also reported in Table 4.1.

For class 1-4, no random perturbation is used and the peak increase parameter ψ increases from 0% to 150%. Note that, increasing the value of ψ, increases the interval of possible travel times, the interval of costs used and the average tail size.

In class 5-8, the peak increase parameter ψ is set to zero and the range of the random perturbation r_ξ increases from 5% to 50%. Note that the random perturbation only affect the costs on the hyperarcs. Hence, the topological structure for hypergraphs in class 5-8, generated with the same seed, are the same. However, the cost interval grows.

The frontier was found in the preliminary tests and phase two was carried out using Rules 5.4.1-5.4.5 with $\varepsilon = \varepsilon_1 = \varepsilon_2 = 0.01$. Hypergraph class 1-8 have also been tested using only Rules 5.4.1 and 5.4.2. Unfortunately, this led to unacceptable CPU times since Rules 5.4.1 and 5.4.2 do not remove enough subhypergraphs from the candidate set to speed up the triangle search significantly.

First, the effect of increasing the peak increase parameter ψ without using a random perturbation is considered (class 1-4). For each class we consider two cases, namely T/C and C/C neg_{cor}. The results are reported in Table 5.2.

The results show that the frontier size grows when the peak increase parameter increases. It is hence relevant to model the peak effect with the TEGP generator. Note also that option C/C neg_{cor} is much harder than T/C, as shown by the maximum values of ε_{ub}, the CPU time and the number of triangles needed to be searched by phase two.

Next we test the effect of changing the random perturbation without a peak effect (class 1 and 5-8). Results are shown in Table 5.3 where five possible values of r_ξ are considered. Like for class 1-4, increasing r_ξ increases the number of extreme nondominated points. Consider the ε_{ub} columns; here average and maximum ε_{ub} fall when the range increases, indicating that the triangles searched become smaller and nondominated points are found closer to the frontier. In the C/C neg_{cor} case this is also reflected in the CPU columns where the CPU time falls. We can conclude that a larger random perturbation gives easier problems. Note that the same does not hold in the peak increase tests, hence

				$\lvert\Phi_\triangle\rvert$		$\text{CPU}_\triangle$		ε_{ub}	
Class	r_ξ	$\lvert\Phi_f\rvert$	$\lvert\triangle\rvert$	ave	max	ave	max	ave	max
T/C									
1	0	4	3	19	59	0.31	0.85	1.80	4.12
5	5	36	3	5	9	0.18	0.70	1.78	4.06
6	10	57	3	5	9	0.21	0.54	1.78	4.01
7	20	85	3	5	21	0.25	0.79	1.82	3.90
8	50	157	3	10	37	0.62	3.32	1.59	3.68
C/C (neg_{cor})									
1	0	7	6	135	690	11.07	113.71	1.55	10.46
5	5	93	13	20	364	2.56	85.99	1.23	10.27
6	10	118	13	15	230	2.08	54.80	1.21	9.90
7	20	171	11	14	78	1.25	31.80	1.21	8.51
8	50	360	11	12	104	1.02	24.48	1.10	2.73

Table 5.3: Preliminary tests – Changing the range of the random perturbation r_ξ (expectation criteria).

Class	9	10	11	12
Grid size	5×8	5×8	10×10	10×10
n	2254	2263	14877	14886
m_h	7383	7405	53220	53241
m_a	80	2262	196	14885
$\lvert T(e)\rvert$	6	6	6	6
H	144	144	288	288
I_T	[3,20]	[3,20]	[3,20]	[3,20]
I_C	[1,2200]	[1,2200]	[1,2200]	[1,2200]
I_W	-	[1,550]	-	[1,550]
Waiting	no	yes	no	yes

Table 5.4: Hypergraph classes 9-12 ($r_\xi = 0.1$ and $\psi = 1$).

it seems that the effect of the random perturbation is most relevant. The preliminary tests show that both parameters ψ and r_ξ must be chosen with caution.

In the remaining tests, the peak increase is set to $\psi = 100\%$, while the range of the random perturbation to $r_\xi = 1\%$. A larger range might result in too easy problems. Four hypergraph classes are considered with two different grid sizes and waiting allowed in two of the classes. The grid size, number of nodes etc after preprocessing are shown in Table 5.4. In all the classes the length of a cycle, peak lengths and off-peak mean travel times and costs are the same as for class 1-8. Note that allowing waiting will make the set of possible strategies between the origin and destination node in G grow significantly. Moreover, using low waiting costs compared to the costs on the hyperarcs may result in a decrease in the expected cost of the hyperpath. That is, we may expect the number of strategies inside a triangle to increase significantly. Therefore, allowing waiting with low

Class		$\lvert\Phi_f\rvert$	CPU_f	$\lvert\Phi_f(\varepsilon)\rvert$	CPU_f	$\lvert\triangle\rvert$	RI_1	RI_2
		$\varepsilon = 0\%$		$\varepsilon = 1\%$				
9	T/C	74	2.61	18	0.38	2	39	98
9	C/C no_{cor}	84	2.98	28	0.62	5	84	106
9	C/C neg_{cor}	186	6.66	63	1.37	11	192	390
10	T/C	98	3.92	23	0.54	2	55	106
10	C/C no_{cor}	140	5.53	33	0.79	5	152	123
10	C/C neg_{cor}	225	8.66	66	1.58	11	231	417
11	T/C	357	72.32	37	3.76	1	69	183
11	C/C no_{cor}	528	106.70	60	6.15	4	173	224
11	C/C neg_{cor}	809	164.02	114	12.19	13	415	711
12	T/C	426	91.83	35	3.80	1	71	185
12	C/C no_{cor}	662	143.06	68	7.51	5	428	240
12	C/C neg_{cor}	923	202.78	117	14.22	14	484	816

Table 5.5: Results phase one (expectation criteria).

waiting costs, will make the problem harder to solve. In the tests of classes 9-12, three types of correlation between the two criteria are considered, namely T/C, C/C no_{cor} and C/C neg_{cor}. We consider phase one and two separately.

Start by looking at the results for phase one, shown in Table 5.5, where both the frontier and an ε-approximation of the frontier are found ($\varepsilon = 1\%$). First, compare the exact results against the approximated ones. Here the number of extreme nondominated points is significantly lower for the approximation, resulting in large savings in CPU time. This implies that the set of large triangles, i.e. the triangles searched by phase two, can be determined much faster by an approximate phase one. Anyway, it must be remarked that the number of large triangles (column $\lvert\triangle\rvert$) is quite limited even compared to the size of the approximated frontier. That is, the frontier contains many points close to each other. In this situation, a decision maker may be satisfied by the results given by phase one, which would make phase two superfluous.

If we compare the different types of correlation, we see that the cases T/C and C/C no_{cor} produce fewer extreme nondominated points than the negatively correlated case C/C neg_{cor}. This is a well-known behavior for the bicriterion shortest path problem, see e.g. Skriver and Andersen [85]. It is not surprising that the T/C option seems to find even fewer points than C/C no_{cor}, since for T/C, the first weight is zero on each hyperarc. Note also that the number of large triangles is highest in C/C neg_{cor} indicating that finding efficient strategies is hardest in this case.

Next, let us consider the exact frontier and compare the different waiting possibilities. For weight option T/C, the number of extreme nondominated points increase when waiting arcs with low costs are used. A possible explanation is that, in a strategy using waiting, the mean travel time will increase (first criterion); however, the mean cost may be smaller. Thus the upper/left point of the criterion space stays the same, while the lower/right point moves to the south-east. This is confirmed by the values RI_j, that are larger when waiting is allowed.

Similar results are obtained when the criteria consider costs; the number of extreme

Class		$\lvert\triangle\rvert$	$\lvert\Phi_\triangle\rvert$			$\text{CPU}_\triangle$		ε_{ub}		$\lvert\Phi_\triangle\rvert$			$\text{CPU}_\triangle$		ε_{ub}	
			U	ave	max	ave	max	ave	max	U	ave	max	ave	max	ave	max
			$\varepsilon_1 = 0.01$							$\varepsilon_1 = 0.1$						
9	T/C	2	0	18	60	4.53	21.80	1.85	3.39	0	4	13	0.09	0.28	1.96	3.39
9	C/C no_{cor}	5	0	9	29	0.28	2.46	1.17	2.66	0	3	10	0.09	0.34	1.65	2.90
9	C/C neg_{cor}	11	0	17	203	1.11	25.80	1.11	2.75	0	6	59	0.14	1.03	1.22	2.76
10	T/C	2	*0	24	216	31.22	116.60	2.01	3.71	0	10	75	11.37	80.28	1.89	3.71
10	C/C no_{cor}	5	0	15	52	1.73	30.28	1.22	2.50	0	5	17	1.39	12.16	1.59	2.89
10	C/C neg_{cor}	11	*0	18	249	7.07	117.73	1.18	4.19	0	7	67	0.92	15.67	1.21	2.75
11	T/C	1	0	1	24	0.16	3.04	0.22	1.49	0	0	6	0.08	1.13	0.23	1.66
11	C/C no_{cor}	4	0	15	132	25.07	300.69	1.12	1.96	0	5	27	1.41	9.84	1.26	2.17
11	C/C neg_{cor}	13	1	34	338	39.65	449.09	1.14	6.48	0	9	149	10.00	356.32	1.10	2.01
12	T/C	1	*0	2	52	20.81	519.10	0.25	2.23	0	3	63	5.24	129.75	0.23	1.78
12	C/C no_{cor}	5	1	94	755	183.19	610.13	1.37	5.15	*0	19	130	28.94	515.62	1.23	2.25
12	C/C neg_{cor}	14	1	35	334	71.55	655.69	1.15	7.24	*0	9	123	19.65	524.27	1.08	3.53

* Few unfinished triangles (below 0.5 on average).

Table 5.6: Results phase two (expectation criteria, use Rule 5.4.5 always).

nondominated points increase when low waiting costs are used. Note that waiting can make the expected cost of both criteria become lower in some cases. However, this does not seem to happen often, as can be seen on the values of RI_j. Using waiting to reduce expected cost on one criterion will in general make the expected cost on the other criterion increase. In graphical terms, the upper/left point of the criterion space moves to the north-west and the lower/right point moves to the south-east resulting in higher values of RI_j.

Consider phase two. An ε-approximation of the frontier was found ($\varepsilon = 1\%$) and phase two applied to the large triangles. At most 10.000 strategies were allowed to be generated for each triangle searched. Two different values of the lower bound ε_1 in Rule 5.4.4 are used to investigate the effect on procedure performance, namely 0.1 and 0.01. Furthermore, recall that using Rule 5.4.5 may result in poor approximations. Therefore it may be relevant to only use Rule 5.4.5 in a triangle when the K best strategies procedure begin to stall. This can be done in the following way.

Assume that, for each strategy $100j$, $j = 1, 2, ...$, we calculate the relative increase in the parametric weight between strategy $100j$ and strategy $100\,(j-1)$ (if $j = 1$ we use the first strategy). Now, if the relative increase is below a specific bound ζ, we use Rule 5.4.5 in the rest of the triangle search. We consider the case where Rule 5.4.5 is applied only when the K best strategies procedure begins to stall, in the experimental tests, with a bound $\zeta = 0.01$. Moreover the case where Rule 5.4.5 is applied at the start of all triangles searched is also considered. In both cases the limit $\varepsilon_2 = 0.01$ is used. Other values of ε_2 have been tested. However, the results is more or less the same.

The results when using Rule 5.4.5 at the start of all triangles searched are reported in Table 5.6, while the results when using Rule 5.4.5, only when the K best strategies procedure begins to stall are reported in Table 5.7.

Consider Table 5.6 first and the results obtained with $\varepsilon_1 = 0.01$. Observe that, in

Class		$\lvert\triangle\rvert$	$\lvert\Phi_\triangle\rvert$			$\text{CPU}_\triangle$		ε_{ub}		$\lvert\Phi_\triangle\rvert$			$\text{CPU}_\triangle$		ε_{ub}	
			U	ave	max	ave	max	ave	max	U	ave	max	ave	max	ave	max
			$\varepsilon_1 = 0.01$							$\varepsilon_1 = 0.1$						
9	T/C	2	0	58	131	27.31	82.28	1.76	3.01	0	10	34	0.88	3.84	1.86	3.22
9	C/C no_{cor}	5	0	42	100	6.33	44.72	1.16	2.66	0	4	11	0.35	1.91	1.62	2.90
9	C/C neg_{cor}	11	*0	58	232	7.98	114.42	1.12	2.75	0	8	55	0.33	3.05	1.20	2.45
10	T/C	2	*0	50	229	41.61	113.39	1.96	3.43	0	15	82	14.84	81.95	1.85	3.52
10	C/C no_{cor}	5	*0	51	110	15.57	107.37	1.15	2.78	0	6	15	3.19	16.99	1.59	2.90
10	C/C neg_{cor}	11	1	59	230	15.79	127.42	1.22	5.42	0	8	54	1.22	15.97	1.20	2.75
11	T/C	1	0	4	82	18.08	449.72	0.23	1.70	0	0	7	0.23	4.66	0.23	1.77
11	C/C no_{cor}	4	*0	43	149	86.08	573.30	1.30	5.45	0	5	25	1.61	11.76	1.20	2.03
11	C/C neg_{cor}	13	1	72	466	85.06	607.14	1.19	8.74	0	10	149	11.73	421.97	1.11	2.01
12	T/C	1	*0	7	97	27.27	516.35	0.26	2.38	0	3	58	6.66	164.99	0.22	1.60
12	C/C no_{cor}	5	2	128	735	248.69	576.30	1.54	7.65	*0	19	132	28.59	505.96	1.22	2.25
12	C/C neg_{cor}	14	2	70	426	107.64	596.87	1.22	8.83	*0	10	123	19.65	517.17	1.09	3.53

* Few unfinished triangles (below 0.5 on average).

Table 5.7: Results phase two (expectation criteria, use Rule 5.4.5 when stall, $\zeta = 0.01$).

general, we find good approximations for most triangles. The average value of ε_{ub} is between 0.22 and 2.01 percent. However, in few triangles poor values of ε_{ub} are obtained. This does not necessarily mean that a poor approximation of the set of nondominated points $\mathcal{W}_{\mathit{eff}}$ is found. Recall that ε_{ub} is an upper bound on the value needed for the approximation to ε-dominate $\mathcal{W}_{\mathit{eff}}$; this upper bound might not be tight in some cases, since ε_{ub} is found by comparing the approximation to the frontier. High values of ε_{ub} may be due to the fact that the true set of nondominated points lies deep inside the triangle.

Clearly, the CPU time grows significantly for the hypergraphs with underlying grid size 10×10. Moreover, CPU time is affected by waiting, since the number of possible strategies increases significantly. Note also that the average CPU time and the maximum CPU time are often quite different. This indicate that the triangles searched may be quite different. On average the triangles are searched relatively fast. However, difficult triangles may appear resulting in high CPU times or even that the procedure stops because the limit of 10.000 strategies are reached.

Now, consider the results obtained with $\varepsilon_1 = 0.1$ in Table 5.6. Here Rule 5.4.4 is stronger and a worse approximation may be expected; this is also the case when no unfinished triangles appear, indeed the average value of ε_{ub} grows. However, the increase in ε_{ub} is small and good savings in CPU time can be obtained. Moreover, if unfinished triangles appear, $\varepsilon_1 = 0.1$ yields better approximations and fewer unfinished gaps. Note also that the number of nondominated points inside a triangle falls. We may argue that a higher value of ε_1 makes the triangle search less selective but faster, and allow us to search "deeper" inside the triangles.

This can be seen in Figure 5.11, where the effect of increasing ε_1 in a difficult triangle is shown for $\varepsilon_1 = 0.01$, 0.1 and 0.2. Here a lot of nondominated points close to the two vertices of the triangle are found, when $\varepsilon_1 = 0.01$ is used, but the search stops ($K = 10.000$) before the whole triangle is searched resulting in large values of ε_{ub}. When

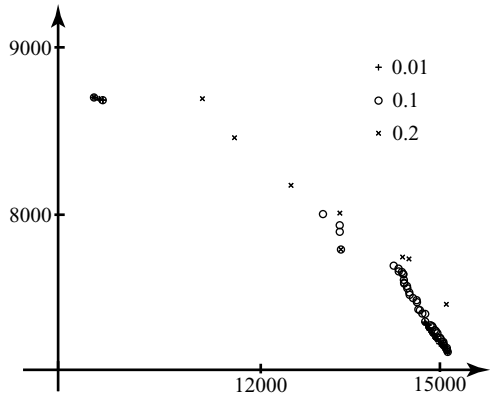


Figure 5.11: Changing ε_1 in a difficult triangle.

increasing ε_1, fewer points are generated, but the triangle is searched deeper, and a better overall approximation is found.

Now, consider Table 5.7 where Rule 5.4.5 is applied only when the K best strategies procedure begins to stall. That is, better approximations are found since Rule 5.4.5 is applied later when a triangled is searched. If we compare the values of ε_{ub} in Table 5.7 with the ones in Table 5.6, only a small decrease in ε_{ub} is obtained. However, the CPU time may grow significantly. In conclusion, applying Rule 5.4.5 later when a triangle is searched does not improve the approximation significantly.

Min-max criteria

Let both criteria be min-max criteria, i.e. we have to find efficient hyperpaths using two distance weighting functions. As pointed out in Section 5.4.2, considering min-max criteria, makes it harder to find efficient strategies, since we cannot find a minimum parametric weight hyperpath by solving a minimum weight hyperpath problem.

In this section, we compare several approximated versions of phase two, based on different settings. For each generated hypergraph, the solutions found with the different settings were merged into a nondominated set Φ_s, representing the best known nondominated set of the problem instance. Since the true frontier in general is not known, we cannot compute an upper bound on the epsilon needed for the nondominated set found to be an ε-approximation of the nondominated set $\mathcal{W}_{\mathit{eff}}$. Instead, we use Φ_s as a benchmark for comparing the relative performance of the various settings.

Only criteria considering cost are used, namely C/C no_{cor} and C/C neg_{cor}. If the first criterion considers time and the second cost, the maximum distance with respect to the time criterion cannot be greater than the time horizon. Thus, efficient solutions can be found by finding the best strategy for different settings of the time horizon. This simpler approach may be much more effective in practice.

First some preliminary tests are performed on smaller hypergraphs where the set of nondominated points $\mathcal{W}_{\mathit{eff}}$ can be found. Hence it is possible to check how good an approximation of the nondominated set we obtain for different ways of

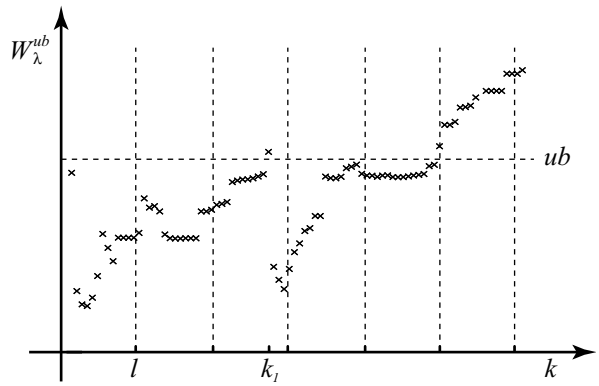


Figure 5.12: Ranking of hyperpaths when procedure *SHTgreedy* is used.

1. adding strategies to the candidate set when a triangle is searched. Recall that two options are possible. We may either add and rank strategies using the weight $\hat{W}_\lambda$ given in Theorem 5.4.7, which is a lower bound on the parametric weight, or we may add strategies using weight W_λ^{ub} returned by using procedure *SHTgreedy* which is an upper bound on the parametric weight.

2. generating the subhypergraphs in the branching tree when a triangle is searched. Recall that we may use either Branching Operation 4.1.2 or the specialized Branching Operation 5.4.1.

Before considering the results of the preliminary tests, let us consider in detail the use of procedure *SHTgreedy* in adding strategies to the candidate set when a triangle is searched. Clearly, procedure *SHTgreedy* does not rank strategies exactly, since procedure *SHTgreedy* does not return the hyperpath with minimum parametric weight. This can be seen in Figure 5.12 where an example of the ranking of the parametric weight for a triangle using procedure *SHTgreedy* is shown. Suppose that the search is stopped as soon as the weight is over the upper bound ub of the triangle ($k = k_1$). In this case, we may miss some (possibly efficient) strategies with weight below ub. This difficulty can be faced as follows. Split the sequence of strategies generated by the K best strategies procedure when searching a triangle into sub-sequences of length l, as shown in Figure 5.12. For each sub-sequence, store the minimum weight W_λ^{ub}, and stop the search if $W_\lambda^{ub} > ub$. Thus, the search may stop after generating $k = l, 2l, 3l, \ldots$ strategies ($k = 6l$ in Figure 5.12). Clearly, higher values of l will in general find better approximations of $\mathcal{W}_{\mathit{eff}}$ on the expense of higher CPU times.

The preliminary tests were carried out on three classes of hypergraphs. The grid size, number of nodes etc after preprocessing are reported in Figure 5.8. A peak increase parameter $\psi = 100\%$ and a range $r_\xi = 10\%$ of the random perturbation is used. No waiting is allowed in class 13-15 which are smaller hypergraphs generated using small grid sizes. The time horizon contains one cycle of 80 time instances, i.e. 6 hours and 40 minutes divided into 5 minutes intervals. Each cycle has two peaks each with a total length of 2 hours; each part of the peak period pk_j has a length of 40 minutes. The first peak starts after 10 minutes ($t = 2$).

Class	13	14	15
Grid size	3×4	4×4	4×5
n	440	497	502
m_h	1111	1392	1454
m_a	51	47	43
$\|T(e)\|$	5	5	5
H	80	80	80
I_T	[3,20]	[3,20]	[3,20]
I_C	[1,2200]	[1,2200]	[1,2200]

Table 5.8: Hypergraph classes 13-15.

Class	$\|\Phi_f\|$	CPU_f	ε_{Φ_s}
13 C/C no_{cor}	2	0.01	4.14
13 C/C neg_{cor}	3	0.02	10.62
14 C/C no_{cor}	3	0.03	5.05
14 C/C neg_{cor}	4	0.05	11.58
15 C/C no_{cor}	3	0.04	1.80
15 C/C neg_{cor}	4	0.05	15.30

Table 5.9: Results frontier approximation (min-max criteria).

The results of phase one are reported in Table 5.9. Here an approximation of the extreme nondominated points is found by using both the weight $\hat{W}_\lambda$ and the weight W^{ub}_λ (see Section 5.4.2). The number of frontier points are small. Furthermore, only a rough approximation of Φ_s is obtained: the ε needed for the approximated frontier to ε-dominate the frontier in Φ_s is between 1.8 and 15.3 percent (column ε_{Φ_s}). However, note that the approximated frontier is only used to guide the search in phase two. If a new extreme nondominated point is found, then the search on the triangles affected is restarted. Finally, note that the values of ε_{Φ_s} are smaller when uncorrelated weights are used, indicating that the uncorrelated case is easier to solve, as expected.

Some of the results for different settings in phase two are reported in Table 5.10. First, consider the results ranking strategies using the lower bound $\hat{W}_\lambda$, Branching Operation 5.4.1 and Rule 5.4.1 and 5.4.2 with $\varepsilon = 0$. Observe that no triangle search is stopped because the limit of 100.000 strategies was reached, except for class 15 in the C/C neg_{cor} case. Hence, due to Theorem 5.4.8, the nondominated set $\mathcal{W}_{\mathit{eff}}$ is found, i.e. Φ_s becomes equal to $\mathcal{W}_{\mathit{eff}}$ resulting in $\varepsilon_{\Phi_s} = 0$. In class 15 (C/C neg_{cor}), a triangle searched did not stop before the limit of 100.000 strategies searched was reached resulting in only an approximation of the nondominated set being found, ($\varepsilon_{\Phi_s} > 0$).

If instead we consider the results ranking strategies using the lower bound $\hat{W}_\lambda$, Branching Operation 4.1.2 and Rule 5.4.1 and 5.4.2 with $\varepsilon = 0$, we see that using Branching Operation 5.4.1 gives much better results. The average and maximum values of ε_{Φ_s} are much higher when using Branching Operation 4.1.2. Furthermore, in all classes we have unfinished triangles and the average number of strategies generated are larger (col-

		$\lvert\Phi_\triangle\rvert$				$\text{CPU}_\triangle$		ε_{Φ_s}	
Class	$\lvert\triangle\rvert$	ave	max	ite	new_f	ave	max	ave	max
	$\hat{W}_\lambda$, B.Operation 5.4.1, Rule 5.4.1 and 5.4.2, $\varepsilon = 0$								
13 C/C no_{cor}	2	1	9	18	1	0.03	0.13	0.00	0.00
13 C/C neg_{cor}	3	2	6	242	1	0.71	5.86	0.00	0.00
14 C/C no_{cor}	3	2	10	158	1	0.34	5.36	0.00	0.00
14 C/C neg_{cor}	4	4	16	4138	6	18.71	139.46	0.00	0.00
15 C/C no_{cor}	2	3	9	5389	2	21.19	288.51	0.00	0.00
15 C/C neg_{cor}	*5	4	12	32101	5	81.91	292.23	0.06	0.80
	$\hat{W}_\lambda$, B.Operation 4.1.2, Rule 5.4.1 and 5.4.2, $\varepsilon = 0$								
13 C/C no_{cor}	*2	1	9	21382	1	11.96	63.06	0.06	1.21
13 C/C neg_{cor}	*3	2	6	18063	1	15.41	92.33	0.00	0.00
14 C/C no_{cor}	*2	1	6	56371	1	70.00	140.97	1.87	10.43
14 C/C neg_{cor}	*3	2	12	81220	4	104.95	148.49	6.74	24.07
15 C/C no_{cor}	*2	2	5	75800	2	170.43	255.61	2.39	11.63
15 C/C neg_{cor}	*3	2	5	98462	3	211.22	260.24	6.15	19.22
	W^{ub}_λ, B.Operation 5.4.1, Rule 5.4.1 and 5.4.2, $\varepsilon = 0.01$, $l = 200$								
13 C/C no_{cor}	2	2	9	32	1	0.11	0.42	0.07	0.53
13 C/C neg_{cor}	3	2	4	106	1	0.38	1.92	0.20	0.88
14 C/C no_{cor}	2	1	7	81	0	0.24	2.28	0.12	0.86
14 C/C neg_{cor}	3	3	14	915	4	2.07	16.34	0.89	7.74
15 C/C no_{cor}	1	2	8	249	1	0.72	2.47	1.44	11.63
15 C/C neg_{cor}	3	4	14	1892	3	6.54	104.46	1.93	10.64
	W^{ub}_λ, B.Operation 5.4.1, Rule 5.4.1, 5.4.2 and 5.4.3, $\varepsilon = 0.01$, $l = 200$								
13 C/C no_{cor}	2	2	9	28	1	0.05	0.24	0.08	0.53
13 C/C neg_{cor}	3	2	4	103	1	0.21	1.09	0.20	0.88
14 C/C no_{cor}	2	1	7	72	0	0.11	1.16	0.12	0.86
14 C/C neg_{cor}	3	3	14	998	4	1.53	15.86	0.76	7.74
15 C/C no_{cor}	1	2	8	246	1	0.61	2.39	1.44	11.63
15 C/C neg_{cor}	3	4	14	3110	3	5.77	90.52	1.84	10.64

* Some unfinished triangles (search max 100.000 strategies).

Table 5.10: Results phase two (min-max criteria).

umn ite). Also Branching Operation 5.4.1 was compared with Branching Operation 4.1.2 when ranking strategies using the upper bound W^{ub}_λ. Here similar results were obtained. Therefore the specialized branching operation for min-max criteria clearly outperforms the general branching operation.

The results using $\hat{W}_\lambda$ and Branching Operation 5.4.1 show that $\mathcal{W}_{\mathit{eff}}$ can be found on small networks. However, using $\hat{W}_\lambda$ on larger networks is not useful. If the grid size grows, too many strategies must be generated before the upper bound of the triangle is reached. This is due to the fact that the lower bound $\hat{W}_\lambda$ may be weak. Instead, we may rank strategies using the upper bound W^{ub}_λ and find an approximation of the nondominated set. Note that the quality of the approximation is affected by the length

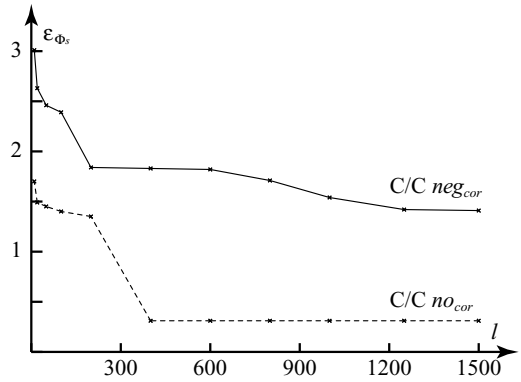


Figure 5.13: Increasing the length l of the sub-sequences (class 15).

of the sub-sequences l and by which rules are used.

Classes 13-15 were tested using the upper bound W_λ^{ub} and Rule 5.4.1 and 5.4.2 with $\varepsilon = 0.01$ for $l = 1$, 10, 20, 50, 100, 200, 400, 600, 800, 1000, 1250 and 1500. Similar tests were performed when Rule 5.4.3 was allowed, too. The results in both cases for $l = 200$ are reported in Table 5.10.

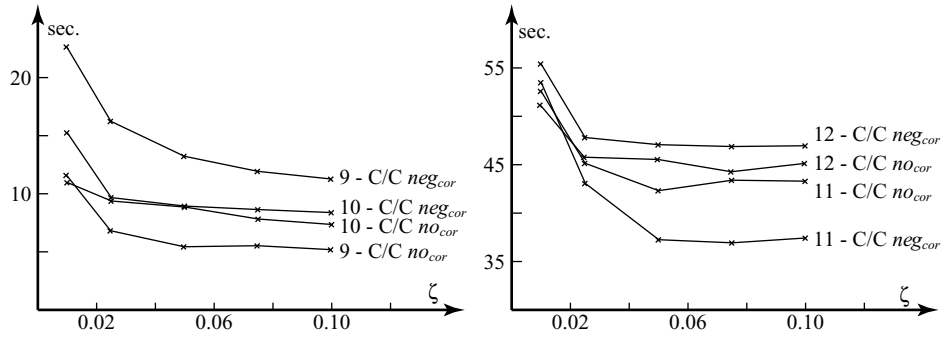
First, note that the average number of strategies generated (column ite) are much smaller compared to when we rank strategies using $\hat{W}_\lambda$ resulting in good savings in CPU. Moreover, we have no unfinished triangles and the average values of ε_{Φ_s} are acceptable.

If we compare the results where Rule 5.4.3 is used, too, we see that the approximation is not worse. However, we obtain savings in CPU by using Rule 5.4.3. That is, removing subhypergraphs from the candidate set where efficient hyperpaths will contain inefficient subhyperpaths, does not seem to affect the approximation significantly.

Increasing the length of the sub-sequences l makes the approximation better which can be seen in Figure 5.13 where $(l, \text{ave } \varepsilon_{\Phi_s})$ has been plotted for class 15 in the case where Rules 5.4.1, 5.4.2 and 5.4.3 are used. For small values of l, we have higher values of ε_{Φ_s} which decreases as l increases. For high values of l, there does not seem to be any improvement in ε_{Φ_s}.

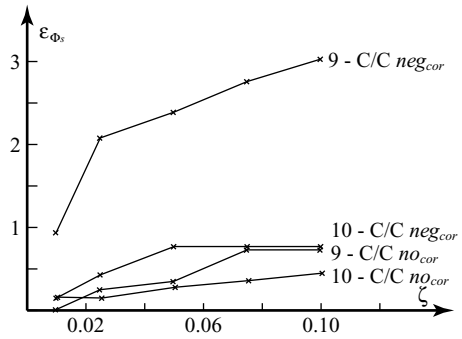
Note also that the C/C neg_{cor} case is harder to solve than the C/C no_{cor} case. We obtain worse approximations and the CPU times are significantly higher in the negatively correlated case.

Now consider classes 9-12 (see Table 5.4 on page 106). Due to the higher grid size, the number of hyperpaths corresponding to points inside an triangle searched by phase two may increase significantly. Hence, even using the upper bound W_λ^{ub} with Rules 5.4.1-5.4.3, may fail to search a full triangle. Therefore Rule 5.4.5 has to be used in these cases. As pointed out in Section 5.4.1, Rule 5.4.5 may result in a poor approximation and should only be used when the K best strategies procedure begins to stall. This can be checked in the following way. Using the length l of the sub-sequences, the lowest and highest parametric weight for each interval is stored. If the percentage increase from the lowest weight to the highest weight is below the bound ζ, then Rule 5.4.5 is applied to the rest of the triangle search. It is expected that a higher bound ζ will result in a worse approximation but better CPU.

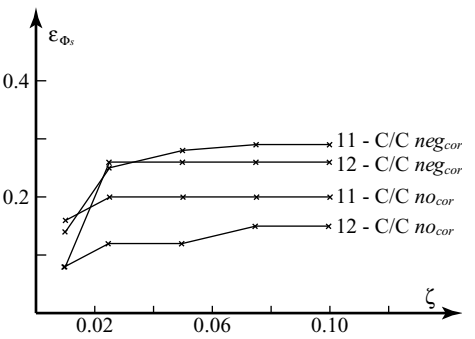


(a) Class 9 and 10.

(b) Class 11 and 12.



(c) Class 9 and 10.



(d) Class 11 and 12.

Figure 5.14: Increasing the bound ζ ($l = 50$).

Classes 9-12 were tested using Branching Operation 5.4.1 and upper bound W_λ^{ub} with Rules 5.4.1-5.4.3 and 5.4.5, $l = 50$ and $\zeta = 0.1$, 0.075, 0.05, 0.025 and 0.01. The results are shown in Figure 5.14 where the bound ζ has been plotted against average CPU and average ε_{Φ_s} for each triangle.

For the (ζ, average CPU) plots, we see that the CPU time only increases for small values of ζ (ζ below 0.05). Note that the increase in the weight for each sub-sequence is compared against the bound ζ for $k = 50, 100, ...$ and hence Rule 5.4.5 can first be applied for $k = 50$. Therefore almost the same CPU time for $\zeta \geq 0.05$ is a result of the fact that Rule 5.4.5 is almost always applied for $k = 50$. That is, we obtain almost the same approximation for all bounds $\zeta \geq 0.05$. This can also be seen in the (ζ, average ε_{Φ_s}) plots, here the values of ε_{Φ_s} are quite stable for $\zeta \geq 0.05$ (except maybe in class 9 C/C neg_{cor}).

For ζ below 0.05 we obtain better values of ε_{Φ_s}, i.e. Rule 5.4.5 is applied later in the triangle search. Note that this may result in a high increase in CPU time.

Finally, we compare the different weight options. As expected correlated weights make the problem harder to solve, we have higher values of ε_{Φ_s} and more triangles must be

Class		$\lvert\Phi_f\rvert$	ite	CPU_f	$\lvert\triangle\rvert$	$\lvert\Phi_f(\varepsilon)\rvert$	ite	CPU_f	$\lvert\triangle\rvert$	RI_1	RI_2
		$\varepsilon = 0$				$\varepsilon = 0.01$					
9	T/C	5	25	0.54	4	5	22	0.48	3	48	94
9	C/C no_{cor}	4	28	0.57	3	4	28	0.57	3	98	103
9	C/C neg_{cor}	8	76	1.45	7	8	76	1.42	7	201	324
10	T/C	35	172	6.01	34	13	55	1.73	4	84	100
10	C/C no_{cor}	41	213	6.74	40	11	50	1.42	4	424	121
10	C/C neg_{cor}	32	190	5.70	31	12	84	2.19	7	201	349
11	T/C	6	84	8.90	5	6	76	7.86	4	36	136
11	C/C no_{cor}	8	216	21.45	7	8	214	20.72	6	140	92
11	C/C neg_{cor}	11	243	25.04	10	11	230	23.38	8	280	365
12	T/C	73	605	96.10	72	15	98	15.31	4	71	155
12	C/C no_{cor}	113	2124	265.71	112	22	341	44.55	7	756	126
12	C/C neg_{cor}	114	2058	259.11	113	24	323	43.73	10	532	541

Table 5.11: Results phase one (expectation criteria).

searched.

Last, we remark that Branching Operation 5.4.1 may have been used in another way. Recall that we create subhypergraphs by considering a valid ordering of the nodes in the maximal weight paths P_1 and P_2 in π backwards (see Section 5.4.2). This results in all hyperpaths in subhypergraph $\bar{\mathcal{H}}^i$ must contain a specific end-tree $\bar{\eta}^i$. However, another ordering can be used as long as the hyperpaths still must contain an end-tree. Therefore we could first branch backwards on the nodes in the valid ordering of P_2 and next do the same for the nodes in the valid ordering of P_1. This approach also provides us with an end-tree.

Branching Operation 5.4.1, as explained in Section 5.4.2 follows a "breadth-first" approach while the above-mentioned approach follows a "depth-first" approach. The "depth-first" approach was also tested on the classes above. However, the best results were obtained using a "breadth-first" approach.

5.6.3 A priori route choice

In this section the problem of finding efficient path-strategies is considered. Our main goal here is to evaluate phase one and phase two under different criteria and different correlation between the two criteria.

The same criteria and correlations as considered under time-adaptive route choice will be considered and the procedures will be tested on hypergraph classes 9-12. Recall that, in class 10 and 12, waiting is allowed. Therefore, as pointed out in Section 5.5.2, the K best path-strategy procedure for searching a triangle may stall. However, an approximation can be found by simply allowing one path-strategy to be generated for each path in G. This approach will be used in all the tests for class 10 and 12.

Class	$\lvert\triangle\rvert$	$\lvert\Phi_\triangle\rvert$ ave	$\lvert\Phi_\triangle\rvert$ max	$\text{CPU}_\triangle$ ave	$\text{CPU}_\triangle$ max	ε_{ub} ave	ε_{ub} max	$\lvert\triangle\rvert$	$\lvert\Phi_\triangle\rvert$ ave	$\lvert\Phi_\triangle\rvert$ max	$\text{CPU}_\triangle$ ave	$\text{CPU}_\triangle$ max	ε_{ub} ave	ε_{ub} max
	$\varepsilon = 0$							$\varepsilon = 0.01$						
9 T/C	4	1	4	0.16	0.33	0.0	0.0	3	1	4	0.14	0.28	1.0	1.0
9 C/C no_{cor}	3	1	4	0.20	0.40	0.0	0.0	3	1	4	0.18	0.34	1.0	1.0
9 C/C neg_{cor}	7	3	11	0.73	2.22	0.0	0.0	7	3	10	0.54	1.86	1.0	1.0
10 T/C	34	0	10	0.12	0.48	0.5	7.3	4	1	3	0.18	0.44	2.9	7.3
10 C/C no_{cor}	40	0	6	0.13	0.60	0.9	9.1	4	1	5	0.22	0.49	4.4	9.1
10 C/C neg_{cor}	31	1	11	0.42	2.86	1.3	8.1	7	3	10	0.72	2.38	4.1	8.1
11 T/C	5	2	8	3.79	22.91	0.0	0.0	4	1	5	2.00	8.14	1.0	1.0
11 C/C no_{cor}	7	3	17	5.76	47.57	0.0	0.0	6	3	16	4.36	37.72	1.0	1.0
11 C/C neg_{cor}	10	6	26	12.08	43.05	0.0	0.0	8	7	23	7.49	25.85	1.0	1.0
12 T/C	60	2	26	3.43	43.05	0.1	4.6	4	1	7	2.46	10.95	2.1	4.6
12 C/C no_{cor}	112	1	18	1.79	58.43	0.2	5.6	7	3	17	5.64	45.96	3.1	5.6
12 C/C neg_{cor}	113	1	26	2.63	56.94	0.3	6.6	10	6	25	8.31	30.50	2.9	6.6

Table 5.12: Results phase two (expectation criteria).

Expectation criteria

Let both criteria be expectation criteria, i.e. we have to find efficient hyperpaths corresponding to a path-strategy using two mean weighting functions.

The results for phase one using procedure *phaseOne* is reported in Table 5.11. Here both the frontier and an ε-approximation of the frontier is found ($\varepsilon = 1\%$).

First, compare the exact results against the approximated ones. In the approximated case the number of extreme nondominated points is significantly lower if waiting is allowed, resulting in savings in CPU time. This is due to the fact that waiting makes the set of possible path-strategies increase significantly. As a result an approximate phase one may identify the large triangles searched by phase two faster, if low waiting costs are used. Furthermore, note that the number of triangles searched by phase two (column $\lvert\triangle\rvert$) is below half the size of the approximated frontier for class 10 and 12 if waiting is allowed. That is, the frontier contains small triangles. Hence a decision maker may be satisfied by the options offered by phase one, which would make phase two superfluous.

If no waiting is allowed, the differences between the exact and the approximated phase is small. However, if we only want to find an ε-approximation, the number of triangles searched by phase two is a bit lower.

If we compare the different types of correlation there does not seem to be a clear pattern of the negatively correlated case C/C neg_{cor} producing more extreme nondominated points. However, note that the number of triangles that must be searched by phase two, when we find an approximation, indicates how many large triangles there are. Therefore the correlated case C/C neg_{cor} is the hardest to solve, since the number of triangles that must be searched by phase two is highest here.

Finally, recall that the number of frontier points under time-adaptive route choice (Table 5.5) is significantly higher than under a priori route choice. Again due to the fact that the number of possible path-strategies is significantly lower than the number of strategies.

Phase two was applied to both the frontier and the ε-approximation of the frontier with $\varepsilon = 1\%$. The results are reported in Table 5.12. No rules are used except that, if waiting is allowed, then we only consider one path-strategy per path in G when a triangle is searched (which might be considered a rule). Hence if all the triangles defined by the frontier are searched and no waiting is allowed, then the set of nondominated points $\mathcal{W}_{\mathit{eff}}$ is found. Moreover, if no waiting is allowed and we only search the large triangles defined by the approximated frontier, we find an ε-approximation of the frontier. On the other hand, if waiting is allowed, we cannot be sure to find all nondominated points in the triangles we search. As a result, we can only report the upper bound ε_{ub} given in equation (5.5).

Consider the classes where no waiting is allowed and the results obtained with $\varepsilon = 0\%$, i.e. all triangles are searched. Observe that the set of nondominated points can be found at smaller CPU times than under time-adaptive route choice (where only an approximation is found). Moreover, if we are satisfied with an ε-approximation, the CPU times can be reduced slightly.

If waiting is allowed, one must be careful when comparing the average results for different values of ε, since the number of triangles searched is considerably different in the two cases. In fact the average results given for $\varepsilon = 0\%$, report average results for all the triangles, while average results given for $\varepsilon = 1\%$, report average results for the large triangles searched by phase two. In general, good approximations of the nondominated points inside a triangle are found. However, in a few large triangles, poor values of ε_{ub} are obtained. Recall that this does not necessarily mean that a poor approximation of the true set of nondominated points is found. High values of ε_{ub} may be due to the fact that the true set of nondominated points lies deep inside the triangle. Note also that, searching only the large triangles ($\varepsilon = 1\%$) will reduce the overall CPU time for phase two because significantly few triangles are searched.

Finally, note that the number of nondominated points seems to be highest in the negatively correlated case C/C neg_{cor}, as under time-adaptive route choice.

Min-max criteria

Let both criteria be min-max criteria, i.e. we have to find efficient hyperpaths corresponding to a path-strategy using two distance weighting functions.

In this section, we compare different settings used in phase two. For each generated hypergraph, the nondominated points found with the different settings were merged into a nondominated set Φ_s, representing the best known nondominated set of the problem instance. We use Φ_s as a benchmark for comparing the relative performance of the various settings.

As under time-adaptive route choice we have two possible ways of finding an approximation of the path-strategy with minimal parametric weight, namely $\hat{W}_\lambda$ giving a lower bound on the parametric weight or W^{ub}_λ providing an upper bound on the parametric weight.

If no waiting is allowed, using $\hat{W}_\lambda$ provides us with a complete method, i.e. the set of nondominated points is found for $\varepsilon = 0$ and an ε-approximation for $\varepsilon > 0$. If waiting is permitted (class 10 and 12), we avoid that the K best path-strategy procedure searching a triangle stalls by allowing only one path-strategy to be generated for each path in G. As a result, only an approximation is found.

Class		$\lvert\triangle\rvert$	$\lvert\Phi_\triangle\rvert$ ave	$\lvert\Phi_\triangle\rvert$ max	$\text{CPU}_\triangle$ ave	$\text{CPU}_\triangle$ max	ε_{Φ_s} ave	ε_{Φ_s} max	$\lvert\Phi_\triangle\rvert$ ave	$\lvert\Phi_\triangle\rvert$ max	$\text{CPU}_\triangle$ ave	$\text{CPU}_\triangle$ max	ε_{Φ_s} ave	ε_{Φ_s} max
		$\hat{W}_\lambda$ and $\varepsilon = 0$							W^{ub}_λ, $\varepsilon = 0.01$ and $l = 30$					
9	C/C no_{cor}	3	1	4	0.32	0.68	0.00	0.00	1	4	1.14	2.19	0.00	0.00
9	C/C neg_{cor}	6	4	14	2.01	5.65	0.00	0.00	4	13	2.16	7.51	0.07	0.41
10	C/C no_{cor}	6	1	7	0.32	1.04	0.57	2.03	1	7	1.61	3.28	0.29	1.30
10	C/C neg_{cor}	9	3	16	2.27	8.52	0.54	1.70	4	16	2.33	6.94	0.34	1.44
11	C/C no_{cor}	7	2	12	17.91	80.12	0.00	0.00	2	12	12.62	30.14	0.37	1.46
11	C/C neg_{cor}	9	7	59	100.42	439.72	0.00	0.00	4	21	20.58	143.57	2.38	5.72
12	C/C no_{cor}	10	2	12	14.92	76.02	0.60	2.91	2	11	9.82	68.13	1.37	4.86
12	C/C neg_{cor}	15	5	35	69.70	385.93	0.46	3.00	3	20	13.96	222.91	2.10	8.19

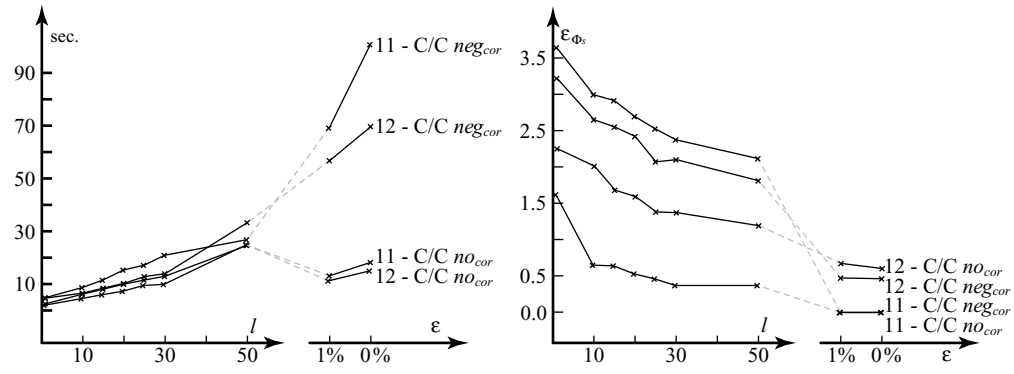
Table 5.13: Results phase two (min-max criteria).

Clearly an approximation is also found, if we use the upper bound W^{ub}_λ. In this case, as under time-adaptive route choice, we can split the sequence of parametric weights of the hyperpaths generated, when a triangle is searched, into sub-sequences of length l. For each sub-sequence we store the minimal parametric weight, and stop the search if the minimal weight is above the upper bound of the triangle.

The procedures were tested using both the lower bound $\hat{W}_\lambda$ with ε equal to 0% and 1% and the upper bound W^{ub}_λ with $\varepsilon = 1\%$ and the length of the sub-sequences l equal to 1, 10, 15, 20, 25, 30 and 50. The results for $\hat{W}_\lambda$ with $\varepsilon = 0$ and W^{ub}_λ with $l = 30$ are given in Table 5.13. By using the lower bound $\hat{W}_\lambda$, the nondominated set can be found for class 9 and 11. Furthermore, using the lower bound $\hat{W}_\lambda$, provides us with the best approximation for class 10 and 12 when comparing to the best known solution Φ_s.

In most cases a good approximation can be found by using the upper bound W^{ub}_λ. However, note that, if the upper bound W^{ub}_λ is used with for instance $l = 30$, then, when searching a triangle, at least 30 iterations must be performed. This is not the case when using the lower bound $\hat{W}_\lambda$ and therefore we may get higher CPU times when using W^{ub}_λ. In fact this is the case for the STD networks with grid size 5×8 (class 9 and 10). Here the number of path-strategies generated when searching a triangle using $\hat{W}_\lambda$ is lower than the number of path-strategies generated when searching a triangle using W^{ub}_λ. Therefore using $\hat{W}_\lambda$ provides us with both the best approximations and the best CPU times. For the STD networks with grid size 10×10 (class 11 and 12), the situation is different. Using $\hat{W}_\lambda$ provides us with the best approximations but using W^{ub}_λ may reduce CPU time significantly.

The average CPU running time and the average value of ε_{Φ_s} for class 11 and 12 is shown in Figure 5.15 for different settings of l (using W^{ub}_λ) and ε (using $\hat{W}_\lambda$). Clearly, increasing l makes the CPU running time grow. However, better approximations are found. If, instead, we use $\hat{W}_\lambda$ for different values of ε, we see that, increasing ε, may reduce CPU running time significantly in the negatively correlated cases (neg_{cor}). In this case the approximation for $\varepsilon = 1\%$ is almost as good as for $\varepsilon = 0\%$.



(a) Class 11 and 12.

(b) Class 11 and 12.

Figure 5.15: CPU time and ε_{Φ_s} for different values of $l = 50$ and ranking.

5.6.4 Summary

This section presents a short summary of the main results of the computational tests under a priori and time-adaptive route choice. We consider expected criteria and min-max criteria separately.

Expectation criteria

First, consider time-adaptive route choice. Here the nondominated set cannot be found since the number of points corresponding to a strategy in the regions we search are huge. Therefore an approximation is found by using rules to prune the candidate set.

The preliminary tests show that the structure of the costs affect the size of the frontier. Increasing the peak increase parameter ψ or the range r_ξ of the random perturbation makes the number of extreme nondominated points grow. A larger frontier may give easier problems.

By using an approximate phase one the set of large triangles, i.e. the triangles searched by phase two, can be determined at lower CPU time. The number of large triangles is quite limited compared to the size of the approximated frontier. That is, the frontier contains many points close to each other. In this situation, a decision maker may be satisfied by the results given by phase one, which would make phase two superfluous.

The CPU time is affected by waiting, since the number of strategies in the regions we search increases significantly.

The average CPU time and the maximum CPU time for searching a triangle are often quite different. This indicate that the triangles searched may be quite different. On average the triangles are searched relatively fast and good approximations are found.

Increasing the value of ε_1 used in Rule 5.4.4 makes the triangle search less selective but faster, and allows us to search "deeper" inside the triangles. Applying Rule 5.4.5 later when a triangle is searched does not improve the approximation significantly.

Second, consider a priori route choice. Here no rules are used except that, if waiting

is allowed, then we only find one path-strategy for each path in G when a triangle is searched. As a result only an approximation is found when waiting is allowed.

If no waiting is allowed the set of nondominated points can be found at smaller CPU times than under time-adaptive route choice (where only an approximation is found). Moreover, if we are satisfied with an ε-approximation, the CPU time can be reduced.

If waiting is allowed an approximation can be determined and in general, good approximations of the nondominated points inside a triangle are found.

Under a priori route choice, the number of frontier points and nondominated points is significantly lower than under time-adaptive route choice.

Finally, note that under both time-adaptive and a priori route choice the number of nondominated points seems to be highest when negatively correlated costs are used. This case is also hardest to solve.

Min-max criteria

For min-max criteria two possible ways of ranking the hyperpaths can be used when a triangle is searched, namely $\hat{W}_\lambda$ giving a lower bound on the parametric weight or W^{ub}_λ providing an upper bound on the parametric weight.

First, consider time-adaptive route choice. Here the nondominated set cannot be found since the number of strategies for the points in the regions we search are huge. Therefore an approximation is found using by rules to prune the candidate set.

The specialized branching operation, Branching Operation 5.4.1, for min-max criteria outperforms the other branching operation.

Using Rule 5.4.3, that is, removing subhypergraphs from the candidate set where efficient strategies will contain inefficient sub-strategies, does not seem to affect the approximation significantly. Applying Rule 5.4.5 later when searching a triangle makes the approximation better. The same holds if we increase the length of the sub-sequences l when using W^{ub}_λ.

The lower bound $\hat{W}_\lambda$ is too weak resulting in a high CPU time for searching a triangle, instead, using W^{ub}_λ provides us with a better approximation and CPU time.

Second, consider a priori route choice. Here no rules are used except that, if waiting is allowed, then we only find one path-strategy for each path in G when a triangle is searched.

The nondominated set can be found if no waiting is allowed by using the lower bound $\hat{W}_\lambda$. Furthermore, using the lower bound $\hat{W}_\lambda$, provides us with the best approximation when waiting is allowed. Note however, ranking hyperpaths using W^{ub}_λ may reduce CPU when considering large grid networks.

As under expectation criteria the number of nondominated points is highest if negatively correlated costs are used. Moreover, the number of nondominated points under time-adaptive route choice is higher than under a priori route choice.

Finally, we point out that the number of nondominated points found under expectation criteria is significant higher than under min-max criteria.

Chapter 6 Further problems in STD networks

Two problems in STD networks are considered in this chapter. Due to time issues, the problems have not been studied as deeply as the problems in the preceding chapters. That is, the results given in this chapter may not be considered as complete. Directions for further research will be pointed out.

In Section 6.1, we consider route choice when the leaving time from the origin is not known. Recall that, in Chapter 4 and 5, strategies S_{o0}, corresponding to leaving the origin o at time zero towards the destination, were considered. However, the leaving time from the origin may not be known, i.e. a set of possible leaving times is given instead.

Consider the MEC or MMC criterion. Here a deterministic cost $c(u, v, t)$ is given for each arc (u, v) and leaving time t. However, the costs may not be known and instead described by a linear function of a parameter λ. The problem now consists in finding the best strategy under time-adaptive route choice for different values of λ. That is, we conduct a parametric analysis where the costs are perturbed along a fixed direction. The problem is considered in Section 6.2. Note that, the same problem is not considered when the MET or MMT criterion is used. Here variations in the travel time $X(u, v, t)$ when leaving node u along arc (u, v) at time t is given by the density of $X(u, v, t)$.

The problem in Section 6.2 may be relevant in intelligent traffic systems where strategies under time-adaptive route choice is calculated online. Here recalculation of the best strategy when the costs change may be avoided by having a parametric solution on hand. The problem may also be considered under a priori route choice. However, it is probably not relevant since decisions are taken a priori, i.e. before we start travelling the route. Therefore the computation time for finding a new best strategy is not as big an issue as under time-adaptive route choice.

6.1 Route choice when the leaving time is not known

Consider an STD network with origin o and destination d. In Chapter 4 and 5, strategies S_{o0} corresponding to leaving the origin o at time zero towards the destination were considered. However, it may be the case that the leaving time from the origin is not known.

Let $L^S(o) \subseteq L(o)$ denote a set of leaving times from the origin satisfying

$$\sum\nolimits_{t \in L^S(o)} \theta_o(t) = 1$$

where $\theta_o(t)$ denotes the probability of leaving the origin at time t. That is,

we leave the origin towards the destination at time t with probability $\theta_o(t) > 0$ for each $t \in L^S(o)$. The problems in Chapter 4 and 5 can now be extended. We here consider strategies S corresponding to leaving the origin o at time $t \in L^S(o)$ towards the destination instead of strategies only leaving the origin at a specific time. More specifically strategy S is defined by

Definition 6.1.1 A strategy S, providing routing choice for travelling from the origin o when leaving at time $t \in L^S(o)$ towards the destination d, is a strategy satisfying

1. $(o,t) \in Dm(S)$, $\forall t \in L^S(o)$
2. No pair (u,t') can be removed from $Dm(S) \setminus \{(o,t) : t \in L^S(o)\}$ such that Definition 3.1.1 still holds.

Consider the criteria introduced in Section 3.2. Given a strategy S satisfying Definition 6.1.1, the expected arrival time when leaving the origin at time $t \in L^S(o)$, is

$$E_T^S(o) = \sum\nolimits_{t \in L^S(o)} \theta_o(t)\, E_T^S(o,t)$$

Moreover, the maximum arrival time is

$$M_T^S(o) = \max\nolimits_{t \in L^S(o)} \{M_T^S(o,t)\}$$

Similarly, if considering cost instead of travel time, we have that the expected cost is given by

$$E_C^S(o) = \sum\nolimits_{t \in L^S(o)} \theta_o(t)\, E_C^S(o,t)$$

and the maximum cost by

$$M_C^S(o) = \max\nolimits_{t \in L^S(o)} \{M_C^S(o,t)\}$$

The problem of finding the best strategy S using one of the criteria introduced in Section 3.2 now consists in finding a strategy with e.g. minimal expected travel time $E_T^S(o)$.

Consider the time-expanded hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ introduced in Section 3.3. Expand hypergraph $\mathcal{H}$ to $\hat{\mathcal{H}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$ by letting

$$\hat{\mathcal{V}} = \mathcal{V} \cup \{\hat{o}\}, \quad \hat{\mathcal{E}} = \mathcal{E} \cup \{(\{o^t : t \in L^S(o)\}, \hat{o})\}$$

That is, we add to $\mathcal{H}$ a dummy node $\hat{o}$ and a dummy hyperarc $\hat{e}$ to $\mathcal{H}$ with $T(\hat{e})$ representing the set of possible leaving times from o and with $h(\hat{e})$ equal to node $\hat{o}$. It is easy to see that Corollary 3.3.2 now can be extended to $\hat{\mathcal{H}}$.

Corollary 6.1.1 *A strategy S, satisfying Definition 6.1.1 providing route choice when leaving node o at time $t \in L^S(o)$ towards d, defines an s-$\hat{o}$ hyperpath π^S in $\hat{\mathcal{H}}$.*

Moreover, using the weights and multipliers given in Lemma 3.3.1, let the weight and multipliers of the new hyperarc $\hat{e}$ added to $\mathcal{H}$ be

$$w(\hat{e}) = 0, \tag{6.1}$$

$$a_e\left(o^t\right) = \theta_o(t), \; o^t \in T(\hat{e}) \tag{6.2}$$

Then it is obvious that the following holds

Theorem 6.1.1 *Given strategy S we have the following*

1. *The expected arrival time $E_T^S(o)$ is equal to the weight of hyperpath π^S using the mean weighting function, weights* (3.6) *and* (6.1) *and multipliers* (3.7) *and* (6.2).
2. *The maximum arrival time $M_T^S(o)$ is equal to the weight of hyperpath π^S using the distance weighting function with weights* (3.6) *and* (6.1).
3. *The expected cost $E_C^S(o)$ is equal to the weight of hyperpath π^S using the mean weighting function, weights* (3.8) *and* (6.1) *and multipliers* (3.7) *and* (6.2).
4. *The maximum cost $M_C^S(o)$ is equal to the weight of hyperpath π^S using the distance weighting function with weights* (3.8) *and* (6.1).

Theorem 6.1.1 implies that the best strategy can be found simply by solving a minimum weight hyperpath problem on $\hat{\mathcal{H}}$. Furthermore, if we consider the problem of finding the K best strategies under time-adaptive route choice in Chapter 4 or the problem of finding efficient strategies under time-adaptive route choice in Chapter 5, then we may consider strategies S where the leaving time is not known simply by considering $\hat{\mathcal{H}}$ and s-$\hat{o}$ hyperpaths π^S in $\hat{\mathcal{H}}$.

The same holds under a priori route choice. Here we find path-strategies satisfying Definition 6.1.1 by finding s-$\hat{o}$ hyperpaths in $\hat{\mathcal{H}}$ corresponding to a path in G.

Note that, since a path-strategy corresponds to a path in G, we here assume that the traveller must follow the same path for all leaving times $t \in L^S(o)$. The problem where just a path must be followed for each leaving time, i.e. the path when leaving the origin at time t may differ from the path when leaving at time $t' \neq t$, seems harder to solve and remains open for further research.

6.2 Route choice when the cost is not known

Let $\mathcal{H}$ be a time-expanded hypergraph with valid ordering $V_{\mathcal{H}} = (s = v_1, ..., v_n = t)$. Assume that, for each hyperarc e, the weight is a nonnegative linear function of the parameter λ

$$w(e, \lambda) = \alpha(e) + \beta(e)\lambda \tag{6.3}$$

where $\alpha(e)$ and $\beta(e)$ are real numbers. That is, the cost of leaving a node u in G, along arc (u, v) at time t, is a linear function of the parameter λ. We assume that $\lambda \in \Lambda = \left[\underline{\lambda}, \overline{\lambda}\right]$, i.e. an interval satisfying $w(e, \lambda) \geq 0$, for all $e \in \mathcal{E}$. Note that, without loss of generality, we may assume that $\underline{\lambda} \geq 0$, since otherwise we may just define $\hat{\lambda} = \lambda - \underline{\lambda}$ and use the parameter $\hat{\lambda}$ instead.

We consider the problem of finding the best strategy S_{o0} for all $\lambda \in \Lambda$ under time-adaptive route choice where S_{o0} defines the route followed when leaving the origin at time zero. According to Corollary 3.3.2 a strategy S_{o0} corresponds to a hyperpath in $\mathcal{H}$. Hence finding the best strategy for a given λ under time-adaptive route choice, corresponds to finding the minimum weight hyperpath between the source s and target t corresponding to leaving the origin o at time zero in hypergraph $\mathcal{H}$ when using the weights $w(e,\lambda)$.

A solution to the problem is a sequence of real numbers $\underline{\lambda} = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{q+1} = \bar{\lambda}$ and s-t hyperpaths $\pi_1, ..., \pi_q$ satisfying that hyperpath π_i is minimal for $\lambda \in [\lambda_i, \lambda_{i+1}]$. Note that the solution is not unique since there may be more than one minimal hyperpath for a fixed λ resulting in different sequences.

We first point out some properties for the mean and distance weighting function. Let $W(\pi,\lambda)$ denote the weight of hyperpath π using weights $w(e,\lambda)$.

Lemma 6.2.1 *Given s-t hyperpath π, the weight $W(\pi,\lambda)$ of hyperpath π, using the mean weighting function, is a continuous linear function of λ, i.e.*

$$W(\pi,\lambda) = a(\pi) + b(\pi)\lambda \tag{6.4}$$

Moreover, the weight $W(\pi,\lambda)$ of hyperpath π, using the distance weighting function, is a continuous piecewise linear convex function of λ, i.e.

$$W(\pi,\lambda) = \begin{cases} a_1(\pi) + b_1(\pi)\lambda & \lambda \in I_1 \\ \vdots & \vdots \\ a_r(\pi) + b_r(\pi)\lambda & \lambda \in I_r \end{cases}$$

with $a_1(\pi) > ... > a_r(\pi)$, $b_1(\pi) < ... < b_r(\pi)$ and $I_i \in \Lambda$, $i = 1, ..., r$.

Proof The weight of hyperpath π using the mean function is according to Theorem 2.3.1

$$\begin{aligned} W(\pi,\lambda) &= \sum_{u \in \mathcal{V}_\pi} f_\pi(u)\left(\alpha(p(u)) + \beta(p(u))\lambda\right) \\ &= \sum_{u \in \mathcal{V}_\pi} f_\pi(u)\,\alpha(p(u)) + \lambda \sum_{u \in \mathcal{V}_\pi} f_\pi(u)\,\beta(p(u)) \end{aligned} \tag{6.5}$$

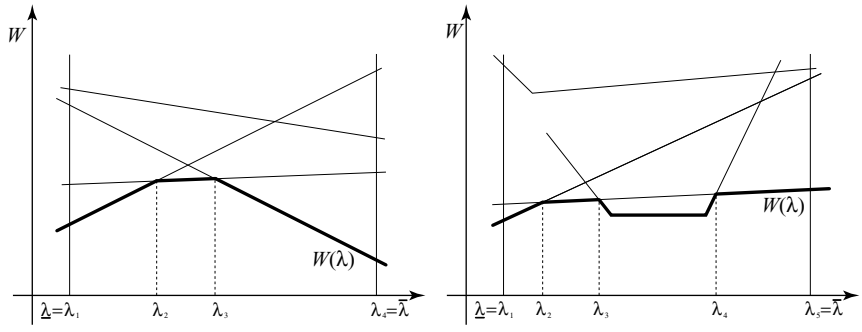
and hence (6.4) holds. Let $\mathcal{P}_\pi$ denote the set of s-t paths contained in π and note that the weight of each path $P \in \mathcal{P}_\pi$ is a linear function $a(P) + b(P)\lambda$. Since the weight using the distance weighting function is equal to the weight of a maximum weight path contained in π we have that

$$W(\pi,\lambda) = \max_{P \in \mathcal{P}_\pi} \{a(P) + b(P)\lambda\}$$

It is well-known that the upper envelope of a set of linear functions defines a continuous piecewise linear convex function. ■

Let $W(\lambda)$ denote the weight of a minimal hyperpath given λ. Using Lemma 6.2.1 we then have

Theorem 6.2.1 *$W(\lambda)$ is a concave piecewise linear function when the mean weighting function is considered and a piecewise linear function when the distance weighting function is considered.*



(a) Mean (b) Distance

Figure 6.1: The weight $W(\lambda)$ for the mean and distance weighting function.

Proof Consider Figure 6.1 where the weight $W(\pi, \lambda)$ has been plotted for all $\pi \in \Pi$ (assuming that $|\Pi| = 4$) for the mean and distance weighting function. The weight $W(\lambda)$ is shown in bold. It is easy to see that the theorem holds. ■

Now, consider expectation criteria where we are interested in finding the best strategy for all $\lambda \in \Lambda$ minimizing the expected travel time or cost. That is, finding a minimum weight *s-t* hyperpath using the mean weighting function. By considering equation (6.5), it is easy to see that $a(\pi)$ is the weight of hyperpath π using hyperarc weights $\alpha(e)$, while $b(\pi)$ is the weight of hyperpath π using hyperarc weights $\beta(e)$. Hence $W(\pi, \lambda)$ is equal to the parametric weight $\gamma(W(\pi), \lambda)$ given in (5.2) using weights $(\alpha(e), \beta(e))$ on each hyperarc. That is, finding $W(\lambda)$ is equivalent to finding the frontier. As a result we can use procedure *phaseOne* shown in Figure 5.4 on page 83 where we, on line 2, replace M with $\bar{\lambda}$ and, on line 3, replace ε with $\underline{\lambda}$.

If, instead, we consider min-max criteria, i.e. we are interested in finding a minimum weight *s-t* hyperpath for all $\lambda \in \Lambda$ using the distance weighting function, the problem is harder. Consider $W(\lambda)$ in Figure 6.1(b), the problem consists in identifying the numbers λ_1 to λ_5 and the minimal hyperpaths π_i with minimum weight $W(\lambda)$ for $\lambda \in [\lambda_i, \lambda_{i+1}]$, $i = 1, ..., 4$. However, $W(\lambda)$ is not concave, i.e. we cannot use a procedure similar to *phaseOne* to solve the problem. Finding an algorithm solving the problem remains open for further research.

Appendix A Hypergraph problems

In this appendix we consider two problems in directed hypergraphs. Both emerged as a result of the study of finding the K best strategies in an STD network. Recall that finding the K best strategies in an STD network corresponds to finding the K minimum weight s-t hyperpaths in the time-expanded hypergraph.

Consider the problem of extending Yen's algorithm, for finding the K shortest loopless paths in a directed graph, to directed hypergraphs. Here hyperarcs are fixed using a forward branching approach, i.e. each hyperpath in the subset $\Pi^{k,i}$ must contain a specific hypertree (see Section 4.1). The problem of finding a minimum weight hyperpath containing a specific hypertree, is considered in Appendix A.1 which is based on Nielsen and Pretolani [71]. It is shown that the problem is $\mathcal{NP}$-hard.

Note that the time-expanded hypergraph is acyclic. Therefore the K minimum weight hyperpaths procedures considered in Chapter 4 may not hold when considering non-acyclic hypergraphs. However, the procedures can with some modifications be extended to non-acyclic hypergraphs. This problem is considered in Appendix A.2. By considering non-acyclic hypergraphs, we do not focus on STD networks anymore, since the time-expanded hypergraph is always acyclic. However, other research areas exist where K minimum weight hyperpaths procedures may be used, as pointed out in the introduction of Chapter 4.

A.1 The subhypertree constrained hyperpath problem

Assume that we are given a weighted hypergraph $\mathcal{H}$, a hypertree $\mathcal{T}$ rooted at s in $\mathcal{H}$, and a node t of $\mathcal{H}$ not in $\mathcal{T}$. The *subhypertree constrained hyperpath* (SCH) problem consists in finding a minimum weight s-t hyperpath containing $\mathcal{T}$ as a subhypertree. We show that this problem is $\mathcal{NP}$-hard, also if $\mathcal{T}$ only contains arcs in $FS(s)$. We consider the distance function here, a simpler construction can be given for the value function. We provide a reduction from the set covering problem (SC), which is well-known to be strongly $\mathcal{NP}$-hard.

An instance of SC is defined by a family $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ of subsets of the set

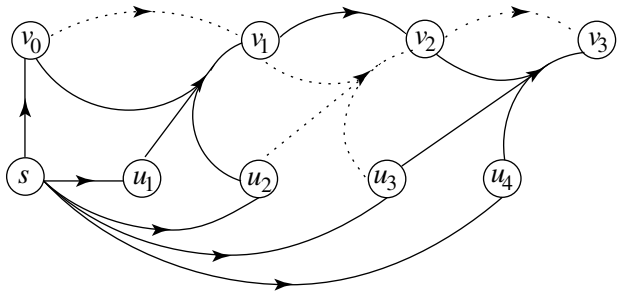


Figure A.1: Hypergraph $\mathcal{H}^C$

$\{1, 2, \ldots, m\}$, where each F_i has a cost c_i. The problem is to find a subset $\mathcal{C} \subseteq \{1, 2, \ldots, n\}$ with minimum cost $c(\mathcal{C}) = \sum_{j \in \mathcal{C}} c_j$ such that

$$\{1, 2, \ldots, m\} = \bigcup_{j \in \mathcal{C}} F_j.$$

Theorem A.1.1 *Problem SCH for the distance function is $\mathcal{NP}$-hard in the strong sense.*

Proof Given an instance of SC, define an instance of SCH as follows. Let $\mathcal{H}^C = (\mathcal{V}^C, \mathcal{E}^C)$ be a weighted hypergraph where

- $\mathcal{V}^C = \{s\} \cup \{u_i : 1 \leq i \leq m\} \cup \{v_j : 0 \leq j \leq n\}$;
- $\mathcal{E}^C = FS(s) \cup \{e^a_j : 1 \leq j \leq n\} \cup \{e^h_j : 1 \leq j \leq n\}$;

here $FS(s)$ contains an arc from s to each node u_i and an arc $(\{s\}, \{v_0\})$; moreover, for each $1 \leq j \leq n$:

- $e^a_j = (\{v_{j-1}\}, v_j)$;
- $e^h_j = (\{u_i : i \in F_j\} \cup \{v_{j-1}\}, v_j)$.

The cost of each hyperarc e^h_j is c_j, arcs have zero costs. Finally, let the hypertree $\mathcal{T}$ contain the arcs in $FS(s)$, and choose the destination node $t = v_n$.

Figure A.1 shows the hypergraph $\mathcal{H}^C$ for an SC instance where $m = 4$, $n = 3$, $F_1 = \{1, 2\}$, $F_2 = \{2, 3\}$ and $F_3 = \{3, 4\}$. The optimal hyperpath is represented by solid lines.

An s-t hyperpath in $\mathcal{H}^C$ is feasible if it contains $\mathcal{T}$. Observe that, in any feasible s-t hyperpath $\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ it is $\mathcal{V}_\pi = \mathcal{V}^C$, and for each $j > 0$, the predecessor $p(v_j)$ is either e^h_j or e^a_j. Therefore π is uniquely defined by the set

$$\mathcal{C} = \{j : e^h_j \in \mathcal{E}_\pi\},$$

and the distance of each node v_j, $j > 0$, is given by the distance of v_{j-1} plus the cost of $p(v_j)$; the distance of π is thus $c(\mathcal{C}) = \sum_{j \in \mathcal{C}} c_j$. Moreover, in a feasible π, each node u_i

must belong to the tail of some hyperarc e_j^h with $j \in \mathcal{C}$. Therefore hyperpath π is feasible for SCH if and only if the corresponding $\mathcal{C}$ is a feasible solution for SC. Since the cost of π is $c(\mathcal{C})$, we conclude that SC reduces to solving the above instance of SCH, and the claim follows. ■

Now consider the forward branching approach in Section 4.1. It is easy to see that finding the minimum weight hyperpath $\pi^{k,i}$ in the set $\Pi^{k,i}$ is a SCH problem and therefore $\mathcal{NP}$-hard.

A.2 Finding the K minimum weight hyperpaths

We consider the problem for finding the K minimum weight hyperpaths in a non-acyclic hypergraph, i.e. we extend the procedures in Section 4.1 to non-acyclic hypergraphs. Note that a hyperpath π is per definition acyclic and hence a valid ordering of π exists. Therefore Branching Operation 4.1.2 and Corollary 4.1.1 still hold. The only difference from the acyclic to the non-acyclic case is that a valid ordering of the hyperpath π^k used in Branching Operation 4.1.2 must be found differently, since no valid ordering is given for the hypergraph. Moreover, the minimum weight hyperpath of each subhypergraph $\mathcal{H}^{k,i}$ must be found differently.

First, consider procedure *K-BS* in Figure 4.5 on page 43. For extending procedure *K-BS* to non-acyclic hypergraphs, procedure *SHT* given in Figure 2.3 on page 13 must be used on line 2, 9 and 14 instead of procedure *SHTacyclic*. Furthermore, a reverse valid ordering of π^k can be found by processing the nodes in π^k from t and backwards.

Extending procedure *K-BSreopt* seems a bit harder. In the non-acyclic case, Theorem 4.1.4 does not hold, that is, the weight in node u_i cannot be found by just processing the hyperarcs in the backward star of node u_i. However, it is easy to see that $W^{k,i}(u_i)$, in Theorem 4.1.4, provides us with a lower bound on the weight in node u_i, i.e. the weight $W^{k,i}(t)$ in Theorem 4.1.6 is a lower bound on the weight of the minimal hyperpath in subhypergraph $\mathcal{H}^{k,i}$.

Procedure *K-BSreopt*, shown in Figure 4.8, can now be extended to non-acyclic hypergraphs. We rank subhypergraphs in the candidate set according to the lower bound $W^{k,i}(t)$ returned by procedure *calcW* on line 16. Since branching tree nodes are ranked according to a lower bound, the minimum weight hyperpath has to be calculated when we pick and remove a branching tree node τ from the heap, i.e. on line 10 the minimum weight hyperpath π_τ in the subhypergraph corresponding to branching tree node τ must be found using procedure *SHT*. Let W_τ denote the weight of hyperpath π_τ. Two cases may happen:

1. The weight W_τ of the minimal hyperpath π_τ in the subhypergraph corresponding to τ is less than or equal to the current minimal lower bound in the candidate set.

2. The weight W_τ of the minimal hyperpath in the subhypergraph corresponding to τ is greater than the current minimal lower bound in the candidate set.

In the first case π_τ is the hyperpath with minimum weight among all the hyperpaths in the candidate set and we output π_τ. In the second case, this is not necessarily true. Therefore we reinsert τ into the candidate set with the lower bound updated to W_τ. That is, we do not perform any branching operation but start a new iteration.

Class	n	m_a	m_h	$\|\tau\|$	reins	CPU_K (reopt)	CPU_K (*K-BS*)	$\|\tau\|$	reins	CPU_K (reopt)	CPU_K (*K-BS*)
				Sum function				Distance function			
1	100	400	5000	9	2.3	1.7	15.1	20	10.7	2.2	31.3
2	300	1200	15000	10	0.5	8.0	78.3	26	1.4	9.4	220.0
3	500	2000	25000	10	1.1	14.5	147.6	28	8.1	17.8	427.0
4	800	3200	40000	10	0.6	24.4	234.0	23	5.0	29.0	606.5
5	1000	4000	50000	9	0.6	31.4	284.2	33	2.1	36.4	1113.4
6	1000	2000	4000	7	0.8	2.4	14.6	16	1.2	2.8	35.8
7	3000	6000	12000	7	0.4	11.1	68.1	18	0.8	12.5	189.4
8	5000	10000	20000	7	0.0	20.4	121.9	16	0.0	22.6	301.8
9	8000	16000	32000	7	0.0	34.9	196.9	14	0.1	38.7	445.3
10	10000	20000	40000	6	0.2	45.0	243.7	14	0.6	50.0	563.4

Table A.1: Sum and distance functions, $K = 500$ (non-acyclic hypergraphs).

Clearly, the modified version on procedure *K-BSreopt* is only effective if the total number of iterations is close to K. That is, the lower bound of the minimal hyperpath in each subhypergraph is tight, resulting in few reinsertions.

The modified versions of procedure *K-BS* and *K-BSreopt* were tested on randomly generated non-acyclic hypergraphs in Nielsen et al. [69] and run on a 700 MHz PIII computer with 512MB RAM using a Linux operating system.

Some of the results are reproduced in Table A.1 where 10 hypergraph classes are considered. Hypergraphs in class 1-5 have a small number of nodes but are dense: the number of arcs is $4n$ and the number of "true hyperarcs" is $50n$. Hypergraphs in class 5-10 have a higher number of nodes but are sparse, the number of arcs is $2n$ and the number of "true hyperarcs" is $4n$. Each row in Table A.1 contains the average results over the five hypergraphs generated for each class. Two weighting functions are considered, namely the sum (see Section 2.3.1) and the distance weighting function.

If we compare the CPU times, we see that the modified version of procedure *K-BSreopt* outperforms the modified version of procedure *K-BS*. Furthermore, the actual number of reinsertions in the modified version of procedure *K-BSreopt* is quite low (10.7% in the worst case) implying that the lower bound is mostly tight. The number of reinsertions tends to be higher for the distance function which is due to the fact that the branching tree size is higher when the distance function is considered.

For further details on finding the K minimum weight hyperpaths in a non-acyclic hypergraphs see Nielsen et al. [69].

Appendix B Basic data structures

In this appendix, basic data structures used by the procedures in this thesis are described. It is assumed that the reader is familiar with object oriented programming and pointers (*ptr*). Examples of program code is written in C++. Objects are written in a box as shown below

Object name	
dt	// data type contained in the object
pDt	// ptr to a data type
...	// other data types

where *dt* is a data type defined inside the object and *pDt* is a pointer to a data type. The prefix *p* before a data type denotes that it is a pointer. Comments to the data types are given with //.

B.1 Data structures for network G and hypergraph $\mathcal{H}$

Consider an STD network with topological network $G = (N, A)$ and time-expanded hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ containing n nodes. The set of hyperarcs $\mathcal{E}$ is divided into *arcs* e and *true hyperarcs* eh (hyperarcs having more than one tail node), that is $\mathcal{E} = (e_1, ..., e_{m_a}) \cup (eh_1, ..., eh_{m_h})$ where m_a and m_h denote the number of arcs and true hyperarcs in $\mathcal{H}$, respectively. Let *tail size ts* denote the total size of the tails of true hyperarcs. A hypergraph/graph structure with a forward and/or a backward representation of $\mathcal{H}/G$ now consists of arrays containing structures representing e.g. an arc in G. The structures are linked together with pointers. For instance a node structure (*NodeH*) corresponding to a node in $\mathcal{H}$ contains

NodeH	
ptr pAFst	// ptr to first arc in BS
ptr pHFst	// ptr to first true hyperarc in BS
ptr pGNode	// ptr to corresponding node in G
...	// other data types

Here *pAFst* and *pHFst* denote pointers to the first arc and to the first true hyperarc in the backward star of the node, respectively. Pointer *pGNode* points to the corresponding node in graph G. Note that we only consider the variables used to link the structures together, other algorithm specific variables will be needed. For instance if minimum weight hyperpaths must be found, we need a weight and a predecessor label in each *NodeH*. Now, consider the backward and forward representation of a hypergraph.

B.1.1 Backward representation of $\mathcal{H}$

Given a node v, a needs to know the hyperarcs belonging to $BS(v)$. We use the node structure given above and define an arc (*ArcH*), true hyperarc (*HArcH*) and a tail (*TailH*) structure.

```
ArcH
pHead  // ptr to head NodeH
pTail  // ptr to tail NodeH
pGArc  // ptr to corresponding arc in G
...    // other data types
```

```
HArcH
pHead   // ptr to head HNode
ppTail  // ptr to first tail ptr
pGArc   // ptr to corresponding arc in G
...     // other data types
```

```
TailH
pTail  // ptr to tail NodeH
...    // other data types
```

The backward representation of a hypergraph can now be obtained by storing the node, arc, true hyperarc and tail structures in arrays. The node array is an array of size $n+2$, where each entry contains a *NodeH* structure.

0	1				n	$n+1$
D	v_1	.	.	.	v_n	D

Here the node structure corresponding to node v_i is stored in entry i. The zero and $n+1$ entries are used as *dummy* (D) nodes so that it is possible to use *for* statements. The nodes are stored such that a valid ordering is $V = (v_1, ..., v_n)$.

The arc array is of size $m_a + 2$, where each entry contains an *ArcH* structure. The array is sorted in a backward star order, i.e. first comes the arcs of $BS(v_1)$, then the arcs of $BS(v_2)$ and so on. Again dummy entries are used.

0	1				m_a	m_a+1
D	e_1	.	.	.	e_{m_a}	D

If *pNode* is a pointer to a $NodeH$, then the backward star of the arcs can now be scanned by using the following *for* statement

```
ptr pANow,pALast;    // ArcH ptr
for(pANow=pNode->pAFst,pALast=(pNode+1)->pAFst;pANow!=pALast;pANow++) {
    statements;
}
```

Here we start with the first arc in the backward star of $pNode$ and scan the arc array until we reach the first arc of the next node. Similarly an true hyperarc array of size $m_h + 2$ containing $HArcH$ structures is made.

0	1				m_h	$m_h + 1$
D	eh_1	.	.	.	eh_{m_h}	D

Again, if $pNode$ is a pointer to a $NodeH$, then the backward star of true hyperarcs can be scanned by using the following *for* statement

```
ptr pHNow,pHLast;    // HArcH ptr
for(pHNow=pNode->pHFst,pHLast=(pNode+1)->pHFst;pHNow!=pHLast;pHNow++) {
    statements;
}
```

To store the tails of the hyperarcs we use a tail array of $TailH$ structures

0	1				ts	$ts + 1$
D	ptr_1	.	.	.	ptr_{ts}	D

The array is sorted in the same way as the hyperarc array. Here the tail nodes of a hyperarc pointed to by $pHArc$ can be scanned using

```
ptr ppNNow,ppNLast;    // ptr to NodeH ptr
for(ppNNow=pHArc->ppTail,ppNLast=(pHArc+1)->ppTail;ppNNow!=ppNLast;pNNow++) {
    pNode=*ppNNow;    // ptr to tail node
    statements;
}
```

B.1.2 Backward and forward representation of $\mathcal{H}$

If a forward representation is needed together with a backward representation then two more arrays are needed – a forward star arc and a forward star true hyperarc array. The $NodeH$ structure must also be modified to contain two more pointers.

NodeH	
ptr pAFst	// ptr to first arc in BS
ptr ppAFst	// ptr to first arc ptr in FS
ptr pHFst	// ptr to first true hyperarc in BS
ptr ppHFst	// ptr to first true hyperarc ptr in FS
ptr pGNode	// ptr to corresponding node in G
...	// other data types

The forward star arc array is of size $m_a + 2$ and contains $ArcH$ pointers.

0	1				m_a	$m_a + 1$
D	ptr_1	.	.	.	ptr_{m_a}	D

Here the forward star of arcs to a node pointed by $pNode$ can be scanned by

```
ptr ppANow,ppALast;    // ptr to ArcH ptr
for(ppANow=pNode->ppAFst,ppALast=(pNode+1)->ppAFst;ppANow!=ppALast;ppANow++) {
    pArc = *ppANow;    // ptr to ArcH
    statements;
}
```

The forward star hyperarc array is of size $ts + 2$ and contains *HArc* pointers.

0	1				ts	$ts+1$
D	ptr_1	.	.	.	ptr_{ts}	D

and if *pNode* is a pointer to a node, then the forward star of hyperarcs can be scanned by using the following *for* statement

```
ptr ppHNow,ppHLast;    // ptr to HArcH ptr
for(ppHNow=pNode->ppHFst,ppHLast=(pNode+1)->ppHFst;ppHNow!=ppHLast;ppHNow++) {
    pHArc = *ppHNow;    // ptr to HArcH
    statements;
}
```

B.1.3 Forward representation of $\mathcal{H}$

If only a forward representation is needed, the backward star arc array can be deleted and the forward star arc array sorted in a forward star order instead. Moreover, the hyperarc array does not have to be sorted in a backward star order.

B.1.4 Representing G and linking G and $\mathcal{H}$

Note G is a special case of a hypergraph, i.e. we can use the same structures as the ones used to represent $\mathcal{H}$. Moreover, since G only contains arcs, we do not need all the arrays and structures used to represent a true hyperarc.

To link $\mathcal{H}$ with G, a pointer for each arc (*ArcH*) and true hyperarc (*HArcH*) in $\mathcal{H}$ is made to the corresponding arc in G (*ArcG*). If a waiting or dummy arc in $\mathcal{H}$ is considered a link to a dummy arc in G is made so that waiting and dummy arcs can be easily identified. Furthermore, for each node in $\mathcal{H}$ (*NodeH*) a pointer is made to its corresponding node structure (*NodeG*) of the node in G which contains

NodeG	
ptr pAFst	// ptr to first arc in BS
ptr ppAFst	// ptr to first arc in FS
ptr ppGList	// ptr to list of NodeH ptr
...	// other data types

Here *ppGList* is used to link G with $\mathcal{H}$. *ppGList* is an array of size time instances + 1 containing pointers to every instance of the node in $\mathcal{H}$. The last entry of the list contains the *null* pointer. The list now can be scanned by using the following *for* statement

```
ptr ppNNow;    // ptr to NodeH ptr
for(ppNNow=ppGList;ppNNow!=null;ppNNow++) {
    pNodeH = *ppNNow;    // ptr to NodeH
    statements;
}
```

The list is ordered in order of time; that is, when scanning the list, we scan the nodes backward according to the valid ordering of $\mathcal{H}$.

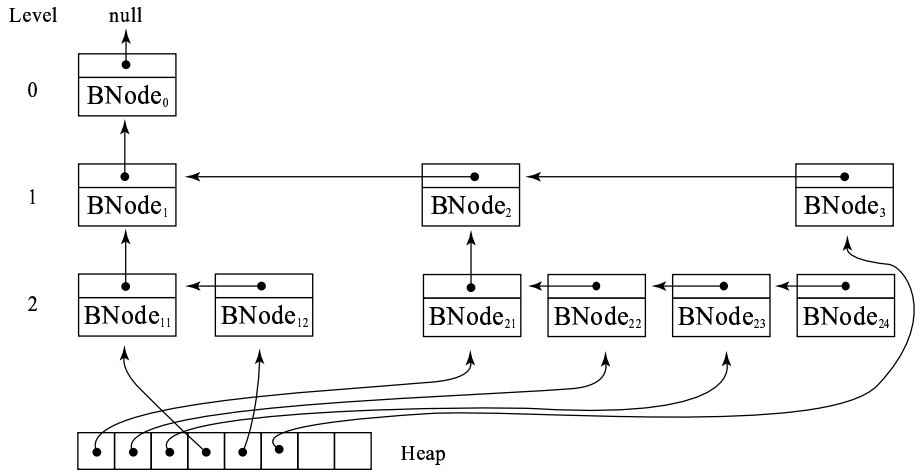


Figure B.1: The branching tree when using Branching Operation 4.1.2, 4.2.1 or 5.4.1.

B.2 Branching tree representation

The branching tree defining the subhypergraphs used in the K best procedures when using Branching Operation 4.1.2, 4.2.1 or 5.4.1 is implemented using a dynamic branching tree of branching tree nodes (*BTNode*) as shown in Figure B.1 which is similar to the branching tree shown in Figure 4.1 on page 38. Each branching node *BTNode* defines a sub(hyper)graph in G (in $\mathcal{H}$) and contains

BTNode	
`flt w`	`// weight/lb of the hyperpath in the subhypergraph`
`ptr pGArc`	`// ptr to arc removed/fixed in G (null otherwise)`
`idx hIdx`	`// idx of the hyperarc we remove, idx<0 if arc`
	`// and idx>0 if true hyperarc (0 otherwise)`
`ptr prev`	`// ptr to the previous BTNode`
`...`	`// other data types`

A heap of branching tree node pointers is used to sort and maintain the candidate set (see Tarjan [87]). If K best strategies procedures are considered, we only branch on hyperarcs in $\mathcal{H}$, i.e. pointer *pGArc* can be removed in all branching tree nodes.

Consider the problem of finding the K best path-strategies using Branching Operation 4.2.1 on π. Note that we create subgraphs of G, if the hyperpath π does not already define a path-strategy. Consider the subgraphs created when using Branching Operation 4.2.1 (see Definition 4.2.1). For subgraph G^i, $i = 1, ..., q-1$, a *BTNode* is created with *pGArc* pointing to the arc we remove and $hIdx = 0$. For subgraph G^q we fix the whole path P_π, i.e. we create a *BTNode* with both $pGArc = null$ and $hIdx = 0$. Moreover, if waiting is allowed and P_π corresponds to an *o-d* path in G, we may use Branching Operation 4.1.2 on the subgraph G^q defined by P_π. In this case we have index $hIdx \neq 0$ and $pGArc = null$. The sub(hyper)graph represented by a branching tree node can now

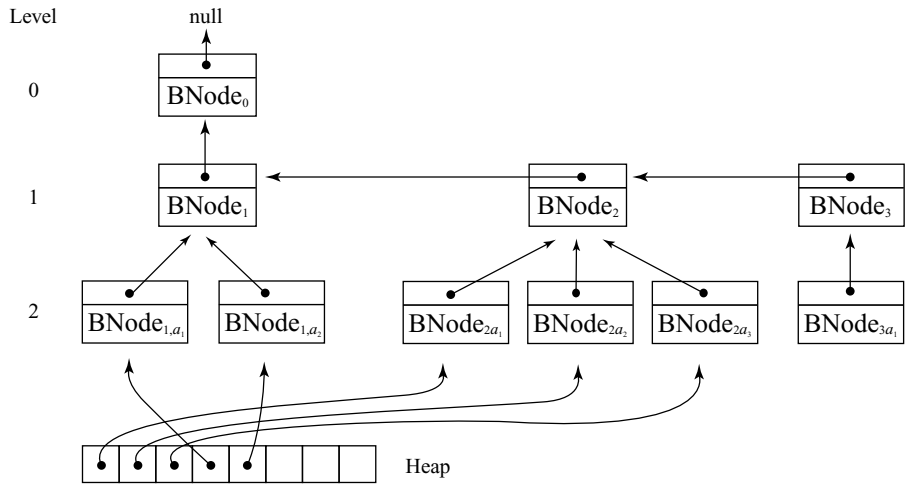


Figure B.2: The branching tree when using Branching Operation 4.2.2.

be built by traversing the unique path from the branching tree node to the root of the branching tree.

When using Branching Operation 4.2.2, i.e. multiple branching, the branching tree is modified as shown in Figure B.2 representing Branching Operation 4.2.2 on a subpath

$$P_\pi = (o = u_1, a_1, u_2, a_2, u_3, a_3, u_4)$$

containing three arcs. $BTNode_1 - BTNode_3$ represent each arc in the subpath. Recall that, for each arc a_i in the subpath with tail u_i, we fix the arcs in $FS(u_i) \setminus \{a_i\}$ one at a time. In the branching tree, this is modelled in the following way

1. For each $BTNode_i$ we create $BTNode_{ia}$ for all $a \in FS(u) \setminus \{a_i\}$ all with $pGArc$ pointing to the arc a we fix.

2. Each $BTNode_{ia}$ points to $BTNode_i$ where $pGArc$ is pointing to the arc a_i in P_π.

The sub(hyper)graph represented by a branching tree node can now be built by traversing the unique path from the branching tree node to the root of the branching tree. Finally, note that, if waiting is allowed and P_π corresponds to an o-d path in G, we may use Branching Operation 4.1.2 on the sub(hyper)graph defined by P_π. In this case the branching tree below the branching tree node we consider will look as shown in Figure B.1.

Appendix C A remark on the definition of a hyperpath

In the last two decades, several problems arising from different application areas have been modelled in terms of *hyperpaths* in *directed hypergraphs*. A general theory of directed hypergraphs was developed for the first time by Gallo et al. [29]. Their paper proposed a definition of hyperpath (called B-path) based on an intuitive concept of *hyperconnection* (called B-connection). The definition aimed at characterizing the topological structure of a minimum weight subhypergraph hyperconnecting a pair of nodes. However, the definition seems to fail in some cases. Here we present a counter-example that satisfies the definition but is not a hyperpath, i.e. it does not hyperconnect two nodes as supposed.

Note that the theoretical results given in [29] are not affected, since they are based on the sound concept of hyperconnection, and do not rely on the definition of a hyperpath. In particular the procedures for finding minimum weight hyperpaths are correct. The same holds true for other papers (e.g. Nguyen, Pretolani, and Markenzon [68] and Pretolani [77]) that adopted the definition given by Gallo et al. [29].

Gallo et al. [29] considered a more general class of directed hypergraphs. The subclass of directed hypergraphs considered in Chapter 2 are denoted B-graphs in Gallo et al. [29]. Basically the difference is that, in the general case, hyperarcs may have more than one head node. Here we only consider B-graphs, i.e. the directed hypergraphs considered in Chapter 2.

C.1 Hypergraphs, hyperconnection, hyperpaths

A *directed hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = (v_1, ..., v_n)$ is the set of *nodes*, and $\mathcal{E} = (e_1, ..., e_m)$ is the set of *hyperarcs*. A hyperarc $e \in \mathcal{E}$ is a pair $e = (T(e), h(e))$, where $T(e) \subset \mathcal{V}$ denotes the set of *tail* nodes and $h(e) \in \mathcal{V} \setminus T(e)$ denotes the *head* node.

A *path* P_{st} in $\mathcal{H}$ is a sequence

$$P_{st} = (s = v_1, e_1, v_2, e_2, ..., e_q, v_{q+1} = t)$$

where, for $i = 1, \ldots, q$, $v_i \in T(e_i)$ and $v_{i+1} = h(e_i)$. A node v is *connected* to node u if a path P_{uv} exists in $\mathcal{H}$. A *cycle* is a path P_{st}, where $t \in T(e_1)$. This is in particular

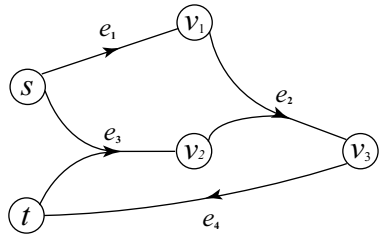


Figure C.1: A counterexample: Hypergraph $\mathcal{H}$.

true if $t = s$. A path is *cycle-free* if it does not contain any subpath which is a cycle, i.e. $v_i \in T(e_j) \Rightarrow j \geq i$, $1 \leq i \leq q+1$. If $\mathcal{H}$ contains no cycles, it is *acyclic*.

The concept of hyperconnection in hypergraphs is captured by the following intuitive definition similar to Proposition 3.1 in Gallo et al. [29].

Definition C.1.1 Hyperconnection to node s in a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$:

1. Node s is hyperconnected to itself;
2. If for some $e \in \mathcal{E}$ all the nodes in $T(e)$ are hyperconnected to s, then node $u = h(e)$ is hyperconnected to s.

The concept of a hyperpath generalizes the notion of simple path in a directed graph. A *hyperpath* from node s to node t in a hypergraph $\mathcal{H}$ is a *minimal* subhypergraph of $\mathcal{H}$ where t is B-connected to s according to Definition C.1.1. Here, minimality is intended with respect to the deletion of nodes and hyperarcs.

A hyperpath can be defined as a sequence of hyperarcs used to prove that t is hyperconnected to s (see e.g. Ausiello et al. [3]). The following topological characterization of a hyperpath, not directly related to Definition C.1.1, has been proposed in Gallo et al. [29].

Definition C.1.2 A *hyperpath* π_{st} from s to t in $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a minimal subhypergraph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ satisfying the following conditions:

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $s, t \in \mathcal{V}_\pi = \bigcup_{e \in \mathcal{E}_\pi} (T(e) \cup H(e))$
3. $u \in \mathcal{V}_\pi \setminus \{s\} \;\Rightarrow\; u$ is connected to s in $\mathcal{H}_\pi$ by means of a cycle-free path.

Unfortunately, Definition C.1.2 is too weak, a counter-example is provided by hypergraph $\mathcal{H}$ in Figure C.1. Hypergraph $\mathcal{H}$ fulfils Definition C.1.2; for example, it contains a cycle-free path from node s to node t, namely $(s, e_1, v_1, e_2, v_3, e_4, t)$. However, according to Definition C.1.1, node t is not hyperconnected to s in $\mathcal{H}_{st}$; the reader can easily check that only node v_1 is hyperconnected to s. Note that $\mathcal{H}$ contains a cycle. Definition C.1.2 can be made correct by further imposing that π_{st} must be acyclic; equivalently, Definition C.1.2 is correct for acyclic hypergraphs (see Property 2.1 in [77]). However, Definition 2.2.1 on page 8 seems to be a more concise and elegant characterization of a hyperpath.

Bibliography

[1] V.G. Adlakha. A Monte Carlo technique with quasirandom points for the stochastic shortest path problem. *American Journal of Mathematical and Management Sciences*, 7(3-4):325–358, 1987.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[3] G. Ausiello, P.G. Franciosa, and D. Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. *Lecture Notes in Computer Science*, 2202:312–328, 2001.

[4] G. Ausiello, G.F. Italiano, and U. Nanni. Optimal traversal of directed hypergraphs. Technical Report TR–92–073, International Computer Science Institute, Berkeley, CA, September 1992.

[5] G. Ausiello, U. Nanni, and G.F. Italiano. Dynamic maintenance of directed hypergraphs. *Theoretical Computer Science*, 72(2–3):97–117, 1990.

[6] J.A. Azevedo, M.E.O.S. Costa, J.J.E.R.S. Maderira, and E.Q. V. Martins. An algorithm for the ranking of shortest paths. *European Journal of Operational Research*, 69:97–106, 1993.

[7] J.A. Azevedo, J.J.E.R.S. Madeira, and E.Q.V. Martins. A computational improvement for a shortest paths ranking algorithm. *European Journal of Operational Research*, 73:188–191, 1994.

[8] C. Berge. *Graphs and hypergraphs*. North-Holland, 1973.

[9] J.R. Birge and F. Louveaux. *Introduction to stochastic programming*. Springer Series in Operations Research. Springer-Verlag, 1997.

[10] J. Brambaugh-Smith and D. Shier. An empirical investigation af some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43:216–224, 1989.

[11] T.H. Byers and M.S. Waterman. Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming. *Operations Research*, 32:1381–1384, 1984.

[12] I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Record*, 1645:170–175, 1998.

[13] H. Chang and U. Lai. Empirical comparison between two k-shortest path methods for the generalized assignment problem. *Journal of Information & Optimization Sciences*, 19(2):153–171, 1998.

[14] Y.L. Chen, D. Rinks, and K. Tang. The first K minimum cost paths in a time-schedule network. *Journal of the Operational Research Society*, 52(1):102 – 108, 2001.

[15] Y.L. Chen and K. Tang. Minimum time paths in a network with mixed time constraints. *Computers & Operations Research*, 25(10):793–805, 1998.

[16] J.C.N. Climaco and E.Q.V. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.

[17] J. Cohen. *Multiobjective Programming and Planning*. Academic Press, New York, 1978.

[18] K.L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14:493–498, 1966.

[19] G.A. Corea and V.G. Kulkarni. Shortest paths in stochastic networks with arc lengths having discrete distributions. *Networks*, 23(3):175–183, 1993.

[20] J.M. Coutinho-Rodrigues, J.C.N. Climaco, and J.R. Current. An interactive bi-objective shortest path approach: Searching for unsupported nondominated solutions. *Computers & Operations Research*, 26:789–798, 1999.

[21] J.R. Current, C.S. ReVelle, and J.L. Cohen. An interactive approach to identify the best compromise solution for two objective shortest path problems. *Computers & Operations Research*, 17(2):187–198, 1990.

[22] N. Deo and C. Pang. Shortest-path algorithms: Taxonomy and annotation. *Networks*, 14:275–323, 1984.

[23] M. Ehrgott. *Multicriteria optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, 2000.

[24] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(4):425–460, 2000.

[25] A. Eiger, P.B. Mirchandani, and H. Soroush. Path preferences and optimal paths in probabilistic networks. *Transportation Science*, 19(1):75–84, 1985.

[26] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2): 653–674, 1999.

[27] H. Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17:583–599, 1969.

[28] G. Gallo, C. Gentile, D. Pretolani, and G. Rago. Max Horn SAT and the minimum cut problem in directed hypergraphs. *Mathematical Programming*, 80:213–237, 1998.

[29] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201, 1993.

[30] G. Gallo and S. Pallottino. Hypergraph models and algorithms for the assembly problem. Technical Report 6, Dipartimento di Informatica, Università di Pisa, March 1992.

[31] G. Gallo and M.G. Scutellà. Minimum makespan assembly plans. Technical Report 10, Dipartimento di Informatica, Università di Pisa, September 1998.

[32] G. Gallo and M.G. Scutellà. A note on minimum makespan assembly plans. *European Journal of Operational Research*, 142(2):309–320, 2002.

[33] M. Garey and D. Johnson. *Computers and Intractability. A Guide of the Theory of NP-Completeness*. W.H. Freeman, 1979.

[34] F. Guerriero and R. Musmanno. Parallel asynchronous algorithms for the K shortest paths problem. *Journal of Optimization Theory and Applications*, 104(1):91–108, 2000.

[35] E. Hadjiconstantinou and N. Christofides. An efficient implementation of an algorithm for finding K shortest simple paths. *Networks*, 34(2):88–101, 1999.

[36] E. Hadjiconstantinou, N. Christofides, and A. Mingozzi. A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest paths relaxations. *Annals of Operations Research*, 61:21–43, 1995.

[37] R.W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[38] G.Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[39] P. Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making, Theory and Application*, number 177 in Lect. Notes Econ. Math. Systems, pages 109–127. Springer-Verlag, Berlin, 1979.

[40] M.I. Henig. The shortest path problem with two objective functions. *European Journal of Operational Research*, 25:281–291, 1986.

[41] J.E. Hershberger, M. Maxel, and S. Suri. Finding the k shortest simple paths: A new algorithm and its implementation. In *Proc. 5th Worksh. Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2003.

[42] P.G Hoel, S.C. Port, and C.J. Stone. *Introduction to Probability Theory*. Houghton Mifflin, 1971.

[43] W. Hoffman and R. Pavley. A method for the solution of the N'th best path problem. *Journal of the Association for Computing Machinery*, 6:506 – 514, 1959.

[44] F. Huarng, P.S Pulat, and L.S. Shih. A computational comparison of some bicriterion shortest path algorithms. *Journal of the Chinese Institute of Industrial Engineers*, 13(2):121–125, 1996.

[45] P. Jaillet. Shortest path problems with node failures. *Networks*, 22(6):589–605, 1992.

[46] R.G. Jeroslow, K. Martin, R.L. Rardin, and J. Wang. Gainfree Leontief substitution flow problems. *Mathematical Programming*, 57:375–414, 1992.

[47] V.M. Jiménez and A. Marzal. A new algorithm for finding the N-best sentence hypotheses in continuous speech recognition. In F. Casacuberta and A. Sanfeliu, editors, *Advances in Pattern Recognition and Applications*, pages 180–187. World Scientific, 1994.

[48] V.M. Jiménez and A. Marzal. Computing the K shortest paths: A new algorithm and an experimental comparison. *Lecture Notes in Computer Science*, 1668:15 – 29, 1999.

[49] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for K shortest simple paths. *Networks*, 12:411–427, 1982.

[50] D.E. Kaufman and R.L. Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *IVHS journal*, 1:1–11, 1993.

[51] E.L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path. *Management Science*, 18(7):401–405, March 1972.

[52] R.P. Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 26:670–676, 1983.

[53] P. Marcotte and S. Nguyen. Hyperpath formulations of traffic assignment problems. In *Equilibrium and advanced transportation modelling*, pages 175–200. Dordrecht: Kluwer Academic Publishers, 1998.

[54] E.Q.V. Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18:123–130, 1984.

[55] E.Q.V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984.

[56] E.Q.V Martins and M.M.B. Pascoal. A new implementation of Yen's ranking loopless paths algorithm. Technical report, Univ. de Coimbra, Portugal, 2000.

[57] E.Q.V. Martins and J.L.E. Santos. A new shortest path ranking algorithm. Technical report, Univ. de Coimbra, 1996.

[58] E.D. Miller-Hooks. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. *Networks*, 37(1):35–52, 2000.

[59] E.D. Miller-Hooks and H.S. Mahmassani. Least possible time paths in stochastic, time-varying networks. *Computers & Operations Research*, 25:1107–1125, 1998.

[60] E.D. Miller-Hooks and H.S. Mahmassani. Optimal routing of hazardous materials in stochastic, time-varying transportation networks. *Transportation Research Record*, 1645:143–151, 1998.

[61] E.D. Miller-Hooks and H.S. Mahmassani. Least expected time paths in stochastic, time-varying transportation networks. *Transportation Science*, 34(2):198–215, 2000.

[62] E.D. Miller-Hooks and H.S. Mahmassani. Path comparisons for a priori and time-adaptive decisions in stochastic, time-varying networks. *European Journal of Operational Research*, 146(1):67–82, 2003.

[63] J. Mote, I. Murthy, and D.L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53:81–92, 1991.

[64] I. Murthy and S. Sarkar. A relaxation-based pruning technique for a class of stochastic shortest path problems. *Transportation Science*, 30(3):220–236, 1996.

[65] S. Nguyen and S. Pallottino. Equilibrium traffic assignment for large-scale transit networks. *European Journal of Operational Research*, 37(2):176–186, 1988.

[66] S. Nguyen and S. Pallottino. Hyperpaths and shortest hyperpaths. In *Combinatorial optimization (Como, 1986)*, volume 1403 of *Lecture Notes in Math*, pages 258–271. Springer, 1989.

[67] S. Nguyen, S. Pallottino, and M. Gendreau. Implicit enumeration of hyperpaths in a logit model for transit networks. *Transportation Science*, 32(1):54–64, 1998.

[68] S. Nguyen, D. Pretolani, and L. Markenzon. On some path problems on oriented hypergraphs. *Theoretical Informatics and Applications*, 32:1 – 20, 1998.

[69] L.R. Nielsen, K.A. Andersen, and D. Pretolani. Finding the k shortest hyperpaths. *To appear in Computers & Operations Research.*

[70] L.R. Nielsen, K.A. Andersen, and D. Pretolani. Bicriterion shortest hyperpaths in random time-dependent networks. *To appear in IMA Journal of Management Mathematics*, 2003.

[71] L.R. Nielsen and D. Pretolani. A remark on the definition of a B-hyperpath. Technical report, Department of Operations Research, University of Aarhus, 2001.

[72] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the Association for Computing Machinery*, 37(3):607–625, 1990.

[73] A. Orda and R. Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3):295–319, 1991.

[74] S. Pallottino and M.G. Scutellà. Shortest path algorithms in transportation models: Classical and innovative aspects. In *Equilibrium and advanced transportation modelling*, pages 245–281. Kluwer Academic Publishers, 1998.

[75] A. Perko. Implementation of algorithms for k shortest loopless paths. *Networks*, 16 (2):149–160, 1986.

[76] M. Pollack. The kth best route through a network. *Operations Research*, 9:578, 1961.

[77] D. Pretolani. A directed hypergraph model for random time-dependent shortest paths. *European Journal of Operational Research*, 123:315–324, 2000.

[78] J.S. Provan. A polynomial-time algorithm to find shortest paths with recourse. *Networks*, 41(2):115–125, 2003.

[79] H.N. Psaraftis and J.N. Tsitsiklis. Dynamic shortest paths in acyclic networks with Markovian arc costs. *Operations Research*, 41(1):91–101, 1993.

[80] K.A. Rink, E.Y. Rodin, and V. Sundarapandian. A simplification of the double-sweep algorithm to solve the k-shortest path problem. *Applied Mathematics Letters*, 13(8):77–85, 2000.

[81] E. Ruppert. Finding the k shortest paths in parallel. *Algorithmica*, 28(2):242–254, 2000.

[82] N.G.F. Sancho. A multi-objective routing problem. *Engineering Optimization*, 10: 71–76, 1986.

[83] C.E. Sigal, A.A.B. Pritsker, and J.J. Solberg. The stochastic shortest route problem. *Operations Research*, 28(5):1122–1129, 1980.

[84] A.J.V. Skriver. A classification of bicriteria shortest path (BSP) algorithms. *Asia-Pacific Journal of Operational Research*, 17:199–212, September 2000.

[85] A.J.V. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest path problems. *Computers & Operations Research*, 27:507–524, sep 2000.

[86] R.E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* Wiley Interscience. Wiley, 1986.

[87] R.E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Conference Series.* SIAM, 1983.

[88] D.M. Topkis. A k shortest path algorithm for adaptive routing in communications networks. *IEEE Transactions on Communications*, 36(7):855–859, 1988.

[89] C. Tung and K.L. Chew. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, 1992.

[90] E.L. Ulungu and J. Teghem. The two-phases method: An efficient procedure to solve biobjective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1994.

[91] M. Visée, J. Teghem, M. Pirlot, and E.L. Ulungu. Two-phases method and branch-and-bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization*, 12:139–155, 1998.

[92] A. Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35(1):70–79, 1987.

[93] J.H. Wu, M. Florian, and P. Marcotte. Transit equilibrium assignment: A model and solution algorithms. *Transportation Science*, 28(3):193–203, 1994.

[94] J.Y. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[95] A.K. Ziliaskopoulos and H.S. Mahmassani. A time-dependent shortest path algorithm for real-time intelligent vehicle/highway system. *Transportation Research Record*, 1408:94–104, 1993.

Index

List of notation

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z