Progress Report May 2001

# A Bicriterion and Parametric Analysis of The Shortest Hyperpath Problem


Lars Relund Nielsen
Department of Operations Research
University of Aarhus
Ny Munkegade, building 530
DK-8000 Aarhus C
Denmark

# A Bicriterion and Parametric Analysis of The Shortest Hyperpath Problem


Lars Relund Nielsen
Department of Operations Research
University of Aarhus
Ny Munkegade, building 530
DK-8000 Aarhus C
Denmark

# Preface

This progress report presents the work of the first two years of my Ph.D. The main fields of my research are hypergraphs, parametric and bicriterion analysis, but also logic has gained some interest. My research activities for the past two years can be summarized as follows:

**Fall 1999:** I started to study the theory of directed hypergraphs. Hypergraphs have different applications, one of them is that hypergraphs can be used to modelling dynamic networks. More precisely a hypergraph model for random time-dependent shortest paths can be formulated. And by finding shortest hyperpaths, we can find the minimum expected/min-max traveltime of the dynamic network. During the semester I followed a study group "Logical inference" concerning how the use of mathematical programming methods can be used in logic inference.

**Spring 2000:** I began the project on bicriterion shortest hyperpaths (bi-SBT) together with my supervisor Kim Allan Andersen. This is a totally new research area, and hence we had to start from scratch by defining a bicriterion hypergraph and formulating the bi-SBT problem. First, I began studying the bicriterion shortest path problem and the methods to solve it. There are two main approaches, namely node labeling and path/tree. The node labeling approach seems to be hard to transfer to hypergraphs because a hyperpath has a more complex structure than a path, resulting in more complex node labeling procedure. Furthermore, an efficient path satisfies that its subpaths are efficient; this is not always the case for hyperpaths. The path/tree approach seems more adaptable to hypergraphs. Most papers use a *k'th* shortest path subprocedure to solve the problem. Therefore I first developed methods to solve the *k'th* shortest hyperpath problem.
During spring 2000 I also began another research project together with Kim Allan Andersen where a parametric analysis of the shortest hyperpath problem is considered. Here we assume that the hyperarc weights are a function of a parameter $\lambda$ and then determine for which values of $\lambda$ a hyperpath is still minimal.

**Fall 2000:** I had a 3 months stay in Camerino, Italy where I visited professor Daniele Pretolani. During my stay, we developed procedures to solve the *k'th* shortest hyperpath problem and the bi-SBT problem. The *k'th* shortest hyperpath problem is solved using a new branching rule which divides the hypergraph into subhypergraphs. The bi-SBT problem is solved using different methods which are tested against each other.
During my stay I also 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 new definition (Manuscript III).

**Spring 2001:** I finished the manuscripts about the bi-SBT problem and the parametric analysis. Wrote this report. I am currently following a study group "Logic-based methods for optimization" about how to use logic-based methods to solve mathematical programming models.

I owe a debt of gratitude to many people who have been crucial to my success in completing the first two years of my Ph.D. First of all, I am thankful to my supervisor Kim Allan Andersen for his guidance, contributions and suggestions. Secondly, I would like to express my gratitude to professor Daniele Pretolani for his contributions and our many discussions about solution methods and implementation details. Special gratitude is also extended to Randi Mosegaard and Ali Khatam for proof reading. Finally, my heartfelt appreciation goes out to Dorthe for her understanding during this writing process.

# Contents



## Manuscripts

# Chapter 1

# Introduction

This progress report presents an overview over the past two years of my Ph.D., and points out directions for further research. The report has tree parts. The first part, which you are reading now, is a brief overview of the contents of the report, i.e. structure and summaries of the manuscripts included in this progress report. The second part is a survey of the main results stated in the three manuscripts included at the back of this report. The second part has the following structure:

**Chapter Two**

Here the basic definitions and properties of directed hypergraphs are stated. Some of the areas are hyperpaths, hypertrees, weighted hypergraphs and the shortest hyperpath problem.

**Chapter Three**

In this chapter we solve the bicriterion shortest hyperpath problem. Section 3.1 presents procedures for finding the $k$ shortest hyperpaths. In Section 3.2 the two-phases method are developed. Finally, Section 3.3 presents a summary of the computational results from Manuscript I.

**Chapter Four**

This chapter contains the temporary results on my project about the parametric shortest hyperpath problem in hypergraphs. We assume that the lengths of the hyperarcs depend on some parameter and present procedures which find the shortest hyperpath from a source node to all other nodes in a hypergraph as a function of the parameter.

**Chapter Five**

In the last chapter, I draw conclusions of my research and point out directions for further research.

Finally the third part contains the original manuscripts which properly should be read separately. Some of the sections in the manuscripts overlap.

# Manuscripts

Three manuscripts are included at the back of this report:

**Manuscript I:** Lars Relund Nielsen, Daniele Pretolani and Kim Allan Andersen, "On bicriterion shortest hyperpaths"

**Abstract**

The bicriterion shortest path problem has been extensively studied in many years. In addition recently there have been a growing focus on dynamic networks including the random time-dependent shortest path problem which can be transformed to the shortest hyperpath problem [29]. But no attempts have been made to find bicriterion dynamic paths.

This paper aims to solve the bicriterion shortest hyperpath problem (bi-SBT). The first step implies development of procedures to solve the *k'th* shortest hyperpath problem in order to pave the way for solving the bi-SBT problem. The paper presents new methods of finding the $k'th$ best solution by branching on lower bounds; methods that may also be applied to digraphs. We next develop different bi-SBT procedures which are tested against each other on randomly generated hypergraphs. The results obtained show that bi-SBT can be solved for large hypergraphs.

**Manuscript II:** Lars Relund Nielsen and Kim Allan Andersen, "Parametric shortest hyperpath problems"

**Abstract**

This paper contains the temporary results on my project about the parametric shortest hyperpath problem in hypergraphs. We assume that the lengths of the hyperarcs depend on some parameter and present procedures which find the shortest hyperpath from a source node to all other nodes in a hypergraph as a function of the parameter. The method extend results known from parametric shortest path problems in directed graphs.

**Manuscript III:** Lars Relund Nielsen, Daniele Pretolani and Kim Allan Andersen, "A remark on the definition of a B-hyperpath"

**Abstract**

In this note we show that a commonly used hyperpath definition for a directed B-hypergraph is wrong. This is done by presenting a counter-example which fulfils the hyperpath definition but which is not a hyperpath.

# Chapter 2

# Directed hypergraphs

This section contains the basic definitions of a directed B-hypergraph, i.e. hypergraphs where each hyperarc only have one node in its head. More general hypergraphs are presented in Gallo, Longo, Pallottino, and Nguyen [13]. In the following a B-hypergraph is referred to as a hypergraph. For a more thorough study and hypergraph examples, see Manuscript I, Section 2.

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

$$e = (T(e), h(e)) \qquad T(e) \subset \mathcal{V} \quad h(e) \subseteq \mathcal{V} \backslash T(e)$$

where $T(e)$ and $h(e)$ denote the *tail* nodes and the *head* node, respectively. The *cardinality* of hyperarc $e$ is the sum of the tail and head nodes, i.e.

$$|e| = |T(e)| + |h(e)| = |T(e)| + 1$$

If $|e| = 2$ hyperarc $e$ is called an *arc*. The *size* of $\mathcal{H}$ is the sum of the cardinalities of its hyperarcs:

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

We denote by

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

the *forward star* and the *backward star* of node $u$, respectively. A *path* $P_{st}$ in a hypergraph $\mathcal{H}$ is a sequence of nodes and hyperarcs in $\mathcal{H}$:

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

where, for $i = 1, ...q + 1, v_i \in T(e_i)$ and $v_{i+1} \in h(e_i)$. 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 $t \in T(e_1)$. A path is *cycle-free* if it does not contain any subpath which is a cycle, i.e.

$$v_i \in T(e_j) \Rightarrow j \geq i \qquad 1 \leq i \leq q + 1$$

If $\mathcal{H}$ contains no cycles, it is *acyclic*. $\tilde{\mathcal{H}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ is a *subhypergraph* of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ if $\tilde{\mathcal{H}}$ satisfies $\tilde{\mathcal{V}} \subseteq \mathcal{V}$ 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 $\mathcal{H}$.

## 2.1 Ordering a hypergraph

We consider a topological ordering of the nodes of a hypergraph.

**Definition 1** Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A *valid ordering*

$$V = (v_{i_1}, v_{i_2}, ..., v_{i_n})$$

of the nodes in $\mathcal{H}$ is a topological ordering of the nodes, such that, for each $e \in \mathcal{E}$ and $u \in T(e)$, node $u$ precedes node $h(e)$ in the ordering (see Figure 2.1).

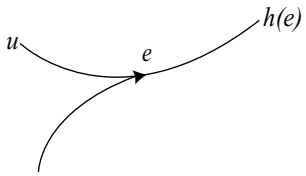



Figure 2.1: A valid ordering $V = (..., u, ..., h(e), ...)$.

It is well-known that $\mathcal{H}$ is acyclic if and only if a valid ordering of the nodes in $\mathcal{H}$ is possible.

## 2.2 Hyperpaths

We here use a slightly different definition of a hyperpath than in Gallo et al. [13] since, in some cases, this definition seems to be not working for B-hypergraphs, see Nielsen, Pretolani, and Andersen [28].
Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. A *hyperpath* $\pi_{st}$ of *origin* $s$ and *destination* $t$, is an acyclic minimal hypergraph (with respect to deletion of nodes and hyperarcs) $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ satisfying the following conditions:

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $s, t \in \mathcal{V}_\pi = \cup_{e \in \mathcal{E}_\pi} (T(e) \cup h(e))$
3. $u \in \mathcal{V}_\pi \setminus \{s\} \Rightarrow u$ is connected to $s$ in $\mathcal{H}_\pi$.

Note that 3. implies that for each $u \in \mathcal{V}_\pi \setminus \{s\}$ there exists a hyperarc $e \in \mathcal{E}_\pi$ such that $h(e) = u$, it follows from the minimality that $e$ is unique. Hyperarc $e$ is called the *predecessor* of $u$ and denoted by $e_\pi(u)$. Note that only a subhypergraph of $\mathcal{H}$ has to be considered when we want to find a hyperpath $\pi_{st}$ because the minimality also implies that the following condition holds:

4. $\exists\ u - t\ path \qquad \forall u \in \mathcal{V}_\pi \setminus \{t\}$

Therefore all nodes which do not have a path to $t$ can be removed from $\mathcal{H}$. We say that node $t$ is *hyperconnected* to $s$ if there exists a hyperpath $\pi_{st}$. Hyperpath $\pi_{st}$ is *different* from hyperpath $\pi_{uv}$ if they do not have the same hyperarcs.

## 2.3 Hypertrees

A *directed hypertree* with *root* $s$ is a hypergraph $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_\mathcal{T})$[1] satisfying the following conditions:

[1] In some definitions it is possible to have more than one root [5].

1. $\mathcal{T}_s$ is acyclic
2. $\{s\} \cap \mathcal{N} = \emptyset$
3. $BS(s) = \emptyset$
4. $|BS(v)| = 1 \quad \forall v \in \mathcal{N}$

A hypertree is the union of hyperpaths to all nodes in $\mathcal{N}$. If $FS(v) = \emptyset$ then $v$ is called a *leaf*. Note that different hypertrees can have the same hyperpath $\pi_{st}$.

## 2.4 Weighted hypergraphs

A *weighted hypergraph* is a hypergraph where each hyperarc $e$ is assigned a real weight $w(e)$. Given a hyperpath $\pi_{st}$ a weighting function $W_\pi$ is a node function which assigns weights $W_\pi(u)$ to all nodes in $\pi_{st}$. The weight of hyperpath $\pi_{st}$ is $W_\pi(t)$. We shall restrict ourselves to *additive weighting functions* which are defined by the recursive equations

$$W_\pi(u) = \begin{cases} w(e_\pi(u)) + F(e_\pi(u)) & u \in \mathcal{V}_\pi \setminus \{s\} \\ 0 & u = s \end{cases}$$

where $F(e)$ is a nondecreasing function of the weights of the nodes in $T(e)$. Two particular weighting functions, namely the *distance* and the *value,* have been studied in detail by Gallo et al. [13], and Pretolani [29] who showed that these two functions have practical applications. The *distance* is obtained by defining $F(e)$ as follows:

$$F(e) = \max_{v \in T(e)} \{W_\pi(v)\}$$

and the *value* is obtained as follows:

$$F(e) = \sum_{v \in T(e)} a_e(v) W_\pi(v)$$

where $a_e(v)$ is a nonnegative multiplier defined for each hyperarc $e$ and node $v \in T(e)$ (see Figure 2.2). The distance (the value) of a hyperpath $\pi_{st}$ is the weight of the hyperpath $\pi_{st}$ with respect to the distance (the value) weighting function.

## 2.5 The shortest hyperpath problem

The *shortest hyperpath problem* (*SBT*)[2] consists in finding the minimum weight hyperpaths from an origin $s$ to all nodes in $\mathcal{H}$ hyperconnected to $s$. In general the problem is hard to solve but if the weighting function is additive, fast algorithms exist. We first define a nondecreasing cycle which ensures that no weight can be decreased through a cycle.

[2]Shortest B-hypertree.

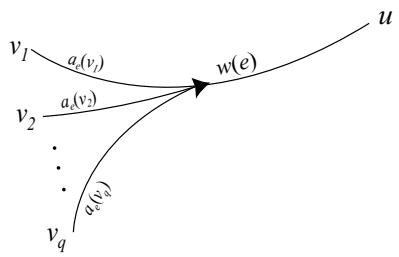



Figure 2.2: The weights and multipliers of the value function $e = (\{v_1, ..., v_q\}, \{u\})$.

**Precondition:** Given hypergraph $\mathcal{H}$ with nondecreasing cycles and nonnegative hyperarc weights, let $W(v_i)$ denote the minimal weight in node $v_i$, $F(e)$ the chosen additive weighting function, $Q$ a candidate set and let $p$ be a predecessor function. Moreover, let the counter $k_j$ for each hyperarc $e_j$ represent the number of nodes in $T(e_j)$ which have been removed from $Q$. Therefore we just update $h(e_j)$ when the weights of all nodes in $T(e_j)$ has been calculated.

**Initialization:** Set $W(v_i) = \infty \ \forall i \in \mathcal{V}$, $k_j = 0 \ \forall e \in \mathcal{E}$, $Q = \{s\}$ and $W(s) = 0$

```
1  while (Q ≠ ∅) do
2      select and remove u ∈ Q;
3      for (e_j ∈ FS(u)) do k_j := k_j + 1
4          if (k_j = |T(e_j)|) then v := h(e_j)
5              if (W(v) > w(e_j) + F(e_j)) then
6                  if (v ∉ Q) then
7                      Q := Q ∪ {v}
8                      if (W(v) < ∞) then
9                          for (e_h ∈ FS(v)) do  k_h := k_h − 1
10                     end if
11                 end if
12                 W(v) := w(e_j) + F(e_j), p(v) := e_j
13             end if
14         end if
15     end for
16 end while
```

Procedure 1: Shortest B-tree procedure (SBT)

**Definition 2** A *nondecreasing cycle* is a cycle $C = \{v_1, e_1, v_2, e_2, ..., v_r, e_r, v_1\}$ that satisfies

$$w(e_r) + F_{v_r}\left[w(e_{r-1}) + F_{v_{r-1}}(...F_{v_2}[w(e_1) + F_{v_1}(z)])\right] \geq z \qquad \forall z \in \mathbf{R}_+ \tag{2.1}$$

here $F_{v_i}(W)$ denotes the function where $v_i \in T(e_i)$ has weight $W$ and all other nodes $u \in T(e_i)$ has weight equal to zero.

That is, if node $v_1$ has temporary weight $z$ then going through $C$ will give no better temporary weight (see Figure 2.3). Now, assume that the weighting function is additive, the weights nonnegative and that all cycles are nondecreasing. Gallo et al. [13] showed that finding the minimum weight hypertree is equivalent to finding a solution to *Bellmans generalized equations*

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

Procedure 1 proposed in [13] finds the minimum weight hypertree containing the shortest hyperpaths. If Dijkstra's principle is used, i.e. select from $Q$ a node $u$ satisfying $W(u) =$

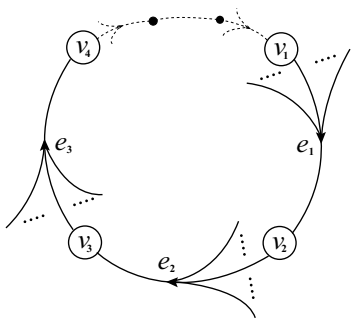



Figure 2.3: A nondecreasing cycle if condition (2.1) holds.

**Precondition:** Given $V$, a valid ordering of $\mathcal{H}$, let $p$ denote the predecessor function and $F(e)$ the chosen additive weighting function.

**Initialization:** Set $W(s) := 0$, $W(v_i) = \infty$ $i = 1, ..., n$

```
1 for (i = 1 to n) do
2     for (e ∈ BS(v_i)) do
3         if (W(v_i) > w(e) + F(e)) then W(v_i) := w(e) + F(e), p(v_i) := e
4     end for
5 end for
```

Procedure 2: Shortest B-tree procedure when $\mathcal{H}$ is acyclic (SBT-acyclic)

$\min\{W(x) \mid x \in Q\}$ at each iteration, the well-known assumption of nonnegative arc weights in standard digraphs becomes

$$w(e) + F(e) \geq W(x) \qquad \forall x \in T(e),\ e \in \mathcal{E} \tag{2.2}$$

that is, the weight in the head must be equal to or larger than the weights in all the nodes in the tail. If assumption (2.2) is satisfied, then Dijkstra's theorem can be extended to hypergraphs.

**Theorem 1** Suppose assumption (2.2) is satisfied and $W(u) = \min\{W(x) \mid x \in Q\}$. 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 and hence line 9 in Procedure 1 can be dropped.

### 2.5.1 Acyclic case

If $\mathcal{H}$ is acyclic, a fast procedure exists [14]. The procedure needs a valid ordering

$$V = (v_0 = s, v_1, ..., v_n)$$

and is stated in Procedure 2. The procedure finds the minimal weight for a node $v$ in one iteration, since the valid ordering assures that the minimal weights in the nodes of $T(e)$ have been found.

# Chapter 3

# The bicriterion shortest hyperpath problem

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 minimizing 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 some real-world 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, where 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). This problem is called the bicriterion shortest path problem. Climaco and Martins [7] solved the problem by first finding an upper bound on one criteria and second use a $k'th$ shortest path procedure to find all efficient paths below that upper bound. In Mote, Murthy, and Olson [26] a two-phases approach was considered. First phase found the supported extreme nondominated points using an LP-relaxation and second phase searched for unsupported nondominated points using a label correcting approach. More recent an interactive approach which finds only a part of the nondominated solutions has been studied [9, 10]. For an overview see Skriver [31].
In this chapter we solve the bicriterion shortest hyperpath problem (bi-SBT). The problem is solved using a $k'th$ shortest hyperpath procedure. Finding the $k'th$ shortest path for digraphs has been extensively studied in recent years [3, 34]. We extend the problem to hypergraphs and present a new fast method for finding all hyperpaths with weight below an upper bound; a method that may also be applied to digraphs. Next, a two-phases method for solving bi-SBT are considered. Here first step finds all supported extreme nondominated points using a NISE like procedure, Cohen [8]. In the second phase, we find all unsupported nondominated points, by searching each triangle defined by the supported extreme nondominated points using a $k'th$ shortest hyperpath procedure.
To keep the size of this chapter to a minimum, we only consider the most efficient methods described in Manuscript I. Manuscript I also contains a section with applications to random time-dependent shortest paths. For more thorough investigation see Manuscript I.

## 3.1 Finding the *k'th* shortest hyperpath

In this section, we describe how to find all shortest hyperpaths, from a root node $s$ to a given destination node $t$, with a weight below an upper bound $ub$. The procedure is an extension of finding the $k$ shortest loopless paths presented in Yen [34]. Lawler [23] presented similar

**Step 1** Use an SBT procedure to find a minimal hyperpath $\pi \in \Pi$.

**Step 2** Divide $\mathcal{H}$ into subhypergraphs $\mathcal{H}_1, ..., \mathcal{H}_q$ which together satisfy: All possible hyperpaths from $s$ to $t$ in $\mathcal{H}$ will be contained in the subhypergraphs except the minimal one found in step 1.

**Step 3** Use an SBT procedure to calculate a minimal hyperpath $\pi_i$ for every subhypergraph $\mathcal{H}_i$.

**Step 4** Pick a hyperpath $\pi_i$ with minimal weight, of the ones calculated in step 3.

Procedure 3: Finding the $2'nd$ shortest hyperpath (main steps)

results in a more general framework where he finds the $k'th$ best solution to a discrete optimization problem. In the following we let $\Pi$ denote the set of hyperpaths $\pi_{st}$ from $s$ to $t$, i.e.

$$\Pi = \{\pi \subseteq \mathcal{H} \,|\pi \text{ has root } s \text{ and destination } t\}$$

The idea of finding the $2'nd$ shortest hyperpath in $\mathcal{H}$ is now as stated in Procedure 3. The subhypergraphs $\mathcal{H}_1, ..., \mathcal{H}_q$ in step 2 can be found by removing and fixing the hyperarcs of the minimal hyperpath $\pi$, as shown in Figure 3.1. Here $\mathcal{V}_\pi = (u_1, ..., u_q)$, and subhypergraph $\mathcal{H}_i$ is obtained from $\mathcal{H}$ by fixing the backward star of node $u_j$ $(1 \leq j < i)$ to $p(u_j)$ and removing $p(u_i)$ from the backward star of node $u_i$. This leads to the following definition.

**Definition 3** Let predecessor function $p$ define the minimal hypertree of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and let an ordering of a subset of nodes $\{u_1, ..., u_l\} \subseteq \mathcal{V}$ be given by

$$V = (u_1, ..., u_l)$$

We say that we *branch on node* $u_i$ meaning that we fix the backward star of node $u_j$ to $p(u_j)$ $(1 \leq j < i)$ and remove $p(u_i)$ from the backward star of node $u_i$. More precisely branching on node $u_i$ corresponds to creating a subhypergraph $\tilde{\mathcal{H}} = \left(\tilde{\mathcal{V}}, \tilde{\mathcal{E}}\right)$ with $\tilde{\mathcal{V}} = \mathcal{V}$ and where the hyperarc set $\tilde{\mathcal{E}}$ is modified in the following way

$$\widetilde{BS}(u_1) = p(u_1), ..., \widetilde{BS}(u_{i-1}) = p(u_{i-1}), \widetilde{BS}(u_i) = BS(u_i) \setminus p(u_i)$$

The ordering $V$ is called *the ordered branching set of* $\mathcal{H}$.

Therefore, if $V$ in Definition 3 contains the nodes of the minimal hyperpath, then subhypergraph $\mathcal{H}_i$ is obtained by branching on node $u_i$.
The SBT procedure finds a minimal hypertree $\mathcal{T}_s$ defined by the predecessor function $p$. Hence, it is possible to find a valid ordering of $\mathcal{T}_s$:

$$V_{\mathcal{T}_s} = (v_1 = s, v_2, ..., v_{n-1}, v_n = t)$$

Note that this valid ordering can be found as the order we pick the nodes in a labelsetting version of procedure SBT, provided that we use Dijkstra's principle and assumption (2.2) is fulfilled. Now if we order the levels of our branching tree in the opposite of $V_{\mathcal{T}_s}$, then by branching on node $u_i$, we only change the minimal weight label in each node which succeeds node $u_i$ in the valid ordering $V_{\mathcal{T}_s}$. The label in all nodes before node $u_i$ in $V_{\mathcal{T}_s}$ will not change. Another good property of using the opposite valid ordering of the minimal hypertree is that we do not have to branch on nodes $v$ in the hyperpath where $|BS(v)| = 1$ in $\mathcal{H}$. This is due to the fact that, if the opposite valid ordering is used, we force the tail nodes of each hyperarc we fix to be nodes in the minimal hyperpath of the subhypergraph. Therefore by removing the hyperarc from a tail node $v$ with $|BS(v)| = 1$, leaves the backward star empty resulting in that no hyperpath $\pi_{st}$ can exist. We now can construct $q$ subhypergraphs $\mathcal{H}_1, ..., \mathcal{H}_q$ which satisfy step 2 in Procedure 3 by using the following branching rule.

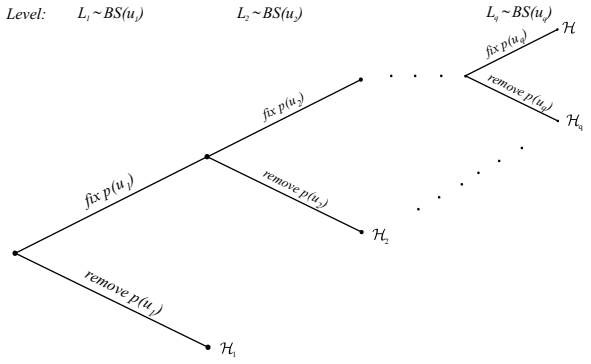



Figure 3.1: The branching tree for the subhypergraphs.

**Branching Rule 1 (Backward branching)** Let predecessor function $p$ define the minimal hypertree $\mathcal{T}_s$ of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ which contains the minimal hyperpath $\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$. Moreover, let $V_{\mathcal{T}_s}$ denote a valid ordering of $\mathcal{T}_s$.

$$V_{\mathcal{T}_s} = (v_1 = s, v_2, ..., v_{n-1}, v_n = t)$$

Now, scan $V_{\mathcal{T}_s}$ backwards from node $v_n$ to node $v_1$ and add node $v_i$ to the ordered branching set $\tilde{V}$ if

$$v_i \in \mathcal{V}_\pi \text{ and } |BS(v_i)| > 1$$

When $V_{\mathcal{T}_s}$ has been scanned, we now have an ordered branching set

$$V = \left(v_{l_1}, ..., v_{l_p}\right)$$

where the last node added from $V_{\mathcal{T}_s}$ is last in $V$. We now create subhypergraph $\mathcal{H}_i$, $1 \leq i \leq p$ by branching on node $v_{l_i}$.

We can now use Branching Rule 1 in step 2 in Procedure 3, and hence step 3 and 4 will find the $2'nd$ shortest hyperpath.
Finding the $k'th$ shortest hyperpath works in the same way. Now simply store all the subhypergraphs and their corresponding minimal hyperpath in a candidate set so that it is possible to calculate new solutions if this minimal hyperpath is used. A procedure is stated in Manuscript I, section 3.

### 3.1.1 Using lower bounds to reduce computation time

In the above, we assumed that $k$ was known and each time we pick a *k'th* shortest hyperpath, the hypergraph which contains the hyperpath is divided into subhypergraphs. But when we search for nondominated solutions in the bicriterion procedures in Section 3.2, $k$ is not known. We search until we reach an upper bound, which may be lowered during the procedure, and then stop. So, if possible we want to make computations as late as possible, i.e. when we pick a hypergraph, we do not want to calculate the minimal hyperpath for each of its subhypergraphs. Instead, we calculate a lower bound $\underline{W}$ of the shortest hyperpath in each subhypergraph and store this solution. It is obviously crucial that this lower bound is as close as possible to the true value and that the computation of a lower bound is faster than calculating the weight of the shortest hyperpath. We now pick the hypergraph $\tilde{\mathcal{H}}$, from the set we have stored, with minimal lower bound and have two cases:

1. If the lower bound is over the upper search bound, we stop because all solutions needed have been found.

**Precondition:** Let $List$ be a candidate set, $ub$ the upper bound, $Out$ the outputlist and let $lb$ denote the lower bound for subhypergraph $\tilde{\mathcal{H}}$. Moreover, let $q$ denote the number of elements in the ordered branching set of $\bar{\mathcal{H}}$.

**Postcondition:** The outputlist $Out$ has been filled with the hyperpaths which have weight under upper bound $ub$.

**Initialization:** Set $lb = 0$, $List = \{(\mathcal{H}, 0)\}$ and $\bar{\mathcal{H}} = \mathcal{H}$

```
1 while (lb ≤ ub) do use an SBT procedure to calculate the minimal hyperpath π of H̄
2     if (W(π) ≤ ub) then Out := Out ∪ {π}
3     for (i = 1 to q) do create subhypergraph H_i using Branching Rule 1.
4         calculate lb_i using Theorem 2
5         if (lb_i ≤ ub) then set List := List ∪ {(H_i, lb_i)}
6     end for
7     pick (H̄, lb) = arg min_{(H̃,l̃b)∈List} {l̃b}
8 end while
```

Procedure 4: Finding all shortest hyperpaths under an upper bound

2. Otherwise we calculate the right minimal weight hyperpath $\pi$ of $\tilde{\mathcal{H}}$ and create subhypergraphs $\mathcal{H}_1, ..., \mathcal{H}_q$ by using Branching Rule 1. Furthermore, if the lower bound for subhypergraph $\tilde{\mathcal{H}}_i$ is under the upper bound we add $\tilde{\mathcal{H}}_i$ to the candidate set, otherwise not.

Note that we here branch on the minimal hyperpath $\pi$ which has minimal lower bound. Therefore we do not find the $k$ shortest hyperpaths in the right order, i.e. we may find the $1'st$, $2'nd$, $4'th$, $3'rd$, $6'th$, .... but this does not matter, if you just want to find all solutions up to a certain upper bound. We could evidently be compelled to calculate the minimal weight hyperpath $\pi$ of a subhypergraph where $W(\pi)$ is over the upper bound, if the lower bound is weak. But if the lower bound is close to the right value, this would be rare. A lower bound can be found using the following theorem

**Theorem 2** Let predecessor function $p$ define the minimal hypertree $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_{\mathcal{T}_s})$ of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and let $W(u)$ denote the optimal weight in each node $u \in \mathcal{N}$. Moreover, let $V = (u_1, ..., u_p)$ denote the ordered branching set found by using Branching Rule 1 and let $\mathcal{H}_i$ denote the subhypergraph of $\mathcal{H}$ found by branching on node $u_i$. Then

$$\underline{W}(u_i) = \min_{e \in BS(u_i) \setminus p(u_i)} F(e) + w(e) \tag{3.1}$$

is a lower bound on the weight of the minimal hyperpath from $s$ to $u_i$ in $\mathcal{H}_i$ where $F(e)$ denotes the chosen weighting function.

**Proof** Let $\underline{e}$ denote the optimal hyperarc of equation (3.1). Since the optimal weight of node $u \in T(\underline{e})$ in $\mathcal{H}_i$ will never be less than the optimal weight of node $u \in T(\underline{e})$ in $\mathcal{H}$, we have that $\underline{W}(u_i)$ using the optimal weights of $\mathcal{H}$, is a lower bound on the actual minimal weight. ∎

We can now calculate the lower bound weight of node $t$ by changing $W(u_i)$ to $\underline{W}(u_i)$ and calculate new weights for the nodes succeeding node $u_i$ in the valid ordering $V_{\mathcal{T}_s}$. Procedure 4 finds all hyperpaths with a weight under an upper bound $ub$. An illustrative example is given in Manuscript I, Example 3.

### 3.1.2 The acyclic case

Assume that $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is acyclic, and hence a valid ordering of the nodes of $\mathcal{H}$ exists

$$V_{\mathcal{H}} = (v_0 = s, v_1, ..., v_q, ..., v_n = t)$$

We here assume wlog that $t = v_n$, since all nodes in a valid ordering of $\mathcal{H}$ above $t$ can be removed. This valid ordering can now be used in Branching Rule 1 and because $\mathcal{H}$ is acyclic, we can find the minimal hyperpath of $\mathcal{H}$ using an acyclic SBT procedure which uses a backward star representation, i.e. no forward representation is needed in the acyclic case. Moreover, in the acyclic case the lower bounds calculated using Theorem 2 become the true minimal weights of the minimal hyperpath. So to calculate the $k'th$ shortest hyperpath, we only have to calculate the shortest hyperpath of $\mathcal{H}$ and afterwards make trivial computations.

## 3.2 Bicriterion shortest hyperpaths

Let $\mathcal{H}$ be a hypergraph where each hyperarc $e$ is assigned two real weights

$$w_i(e) \qquad i = 1, 2$$

and let $W_i(\pi)$ denote the corresponding additive weighting function of hyperpath $\pi$ using weights $w_i(e)$. Moreover, if the value function is considered, we define nonnegative multipliers for each hyperarc $e$ and node $v \in T(e)$

$$a_i^e(v) \qquad i = 1, 2,\ v \in T(e)$$

In this paper we assume that $a_1^e(v) = a_2^e(v)$, i.e. the multipliers are the same for both weighting functions. Let $\Pi$ denote the set of hyperpaths $\pi_{st}$ from $s$ to $t$, i.e.

$$\Pi = \{\pi \subseteq \mathcal{H} \,|\pi \text{ has root } s \text{ and destination } t\}$$

We now wish to solve *the bicriterion shortest hyperpath problem* (bi-SBT)

$$\min_{\pi \in \Pi} W(\pi) = (W_1(\pi), W_2(\pi)) \tag{3.2}$$

That is to find hyperpaths from a given root $s$ to a given node $t$, where the two weights are minimal in the sense that we cannot improve one weight without worsening the other. We consider the following cases of weighting functions: value/value and distance/distance. Let us follow the terminology in Skriver [31]. Given a hyperpath $\pi$ we say

**Definition 4** A hyperpath $\pi \in \Pi$ is *efficient* if and only if

$$\nexists path\ \tilde{\pi} \in \Pi : W_1(\tilde{\pi}) \leq W_1(\pi) \text{ and } W_2(\tilde{\pi}) \leq W_2(\pi) \tag{3.3}$$

with at least one strict inequality. Otherwise $\pi$ is *inefficient*.

Efficient hyperpaths are defined in decision space and their counterpart is points in criterion space

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

**Definition 5** A point $W(\pi) \in \mathcal{W}$ is a *nondominated* criterion point if and only if $\pi$ is an efficient hyperpath. Otherwise $W(\pi)$ is a *dominated* criterion point.

The criterion points can be partitioned into two kinds, namely supported and unsupported. The supported ones can be further subdivided into supported extreme and supported nonextreme. Let us define

$$\begin{aligned}
\Pi_{Eff} &= \{\pi \in \Pi |\ \pi \text{ is efficient}\} \\
\mathcal{W}_{Eff} &= \left\{W(\pi) \in \mathbb{R}^2 \mid \pi \in \Pi_{Eff}\right\} \\
\mathcal{W}^{\geq} &= conv\left(\mathcal{W}_{Eff} \oplus \left\{\mathbf{w} \in \mathbf{R}^2 \mid \mathbf{w} \geq 0\right\}\right)
\end{aligned}$$

**Definition 6** $W(\pi) \in \mathcal{W}_{Eff}$ is a *supported* nondominated criterion point if $W(\pi)$ is on the boundary of $\mathcal{W}^{\geq}$ denoted $\mathcal{W}^{=}$. Otherwise $W(\pi)$ is an *unsupported* nondominated criterion point.

Notice that unsupported nondominated criterion vectors are dominated by a convex combination of other nondominated criterion vectors [32].

**Definition 7** A supported point $W(\pi)$ is a *supported extreme* nondominated criterion point if $W(\pi)$ is an extreme point of $\mathcal{W}^{\geq}$. Otherwise $W(\pi)$ is an *supported nonextreme* nondominated criterion point.

It is well-known that a set of nondominated points $\Phi = \left\{W^1, W^2, ..., W^l\right\}$ can be ordered in the following way.

$$W_1^1 < W_1^2 < ... < W_1^l \qquad W_2^1 > W_2^2 > ... > W_2^l$$

We call $\Phi$ an *ordered nondominated set*. It is easy to check if a new point $W$ is dominated in $\Phi$, see e.g. Manuscript I, Procedure 10, here referred to as procedure *insert*$(W, \Phi)$.

**Example 1** The criterion space is illustrated in Figure 3.2. Here $\mathcal{W}^{=}$ is the border drawn with the hard lines and $\mathcal{W}^{\geq}$ is the union of $\mathcal{W}^{=}$ and the area above. $W^1$ is the point which has minimal weight w.r.t. weight two when weight one is fixed to its minimal weight. We call $W^1$ the *upper/left point* and likewise $W^8$ the *lower/right point* of the criterion space. All nondominated points will be inside the square defined by $W^1$ and $W^8$, therefore $W^9$ is dominated. The points $W^2, W^3$ and $W^6$ are all supported nondominated points. Furthermore, $W^2$ and $W^6$ are supported extreme nondominated points. All supported extreme points define the *triangles* also called gaps (dashed lines) in which it is possible to find nondominated points. Therefore all points, e.g. $W^7$, outside the triangles will be dominated. If we look at the triangle defined by $W^2$ and $W^6$ we have that $W^4$ is a nondominated point and $W^5$ is dominated. ■

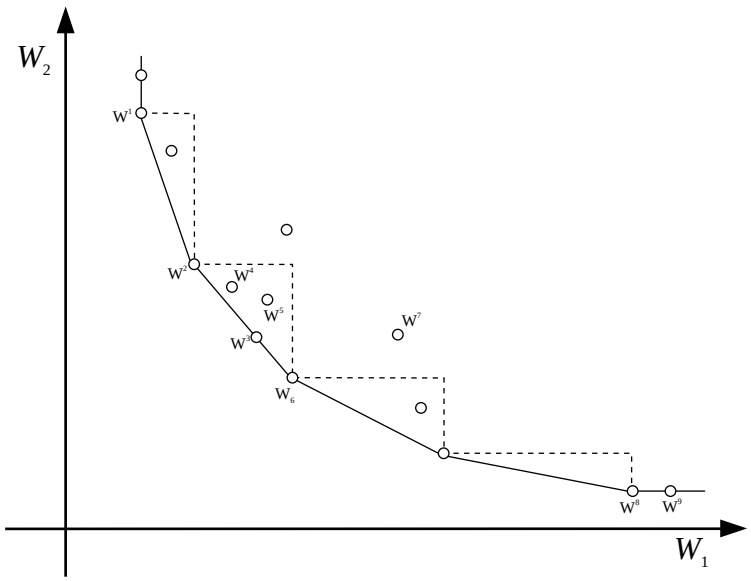



Figure 3.2: Criterion space for different hyperpaths.

### 3.2.1 Finding efficient hyperpaths using a two-phases approach

In this section we consider a two-phases approach where the search for nondominated points are split into two phases. Phase one finds supported extreme nondominated points defining the triangles in which nonsupported nondominated points may be found. Phase two then searches the triangles using a $k'th$ best procedure.

**Phase 1: Finding supported nondominated solutions**

We present a NISE like algorithm [8] for finding supported extreme nondominated points. These can be found by parametrizing the criteria vector. Let $f : (\Pi, \mathbb{R}_+) \longrightarrow \mathbb{R}_+$ denote the function

$$f(\pi, \lambda) = W_1(\pi)\lambda + W_2(\pi)$$

Since the number of hyperpaths in $\Pi$ is finite, we have that $f(\pi, \lambda)$ for fixed $\pi$ defines a line with slope $W_1(\pi)$, intersection $W_2(\pi)$, and the number of lines is finite. Using the parametric function $f(\pi, \lambda)$ we wish to solve

$$f(\lambda) = \min_{\pi \in \Pi} f(\pi, \lambda) \tag{3.4}$$

i.e. we want to find a minimum weight hyperpath $\pi_\lambda = \arg\min_{\pi \in \Pi} f(\pi, \lambda)$ given a fixed $\lambda$. We illustrate this approach with the following example

**Example 2** The criterion space and its corresponding parametric space is shown in Figure 3.3. Here each point $W(\pi)$ corresponds to a line with slope $W_1(\pi)$ and intersection $W_2(\pi)$. For each fixed $\lambda$ we have a minimal line and the lower envelope of the lines defines $f(\lambda)$ which is a nondecreasing piecewise linear function with breakpoints $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. Each piece of $f(\lambda)$ corresponds to a supported extreme nondominated point and each breakpoint $\lambda_i$ corresponds to a value where the two adjacent supported extreme nondominated points have the same minimal parametric weight. If e.g. $\lambda = \lambda_2$ then $W^2$ and $W^3$ have same minimal parametric weight, i.e. fixing $\lambda$ to $\lambda_2$ corresponds to searching for the first point in the direction of the normal of the line shown in Figure 3.3. Furthermore, we have that $W^1$, which is a supported nonextreme nondominated point, touches $f(\lambda)$ in the breakpoint of its adjacent supported extreme points. Note that the whole set of nondominated points is normally nonconvex, i.e. when using the parametric method, we only find points on the

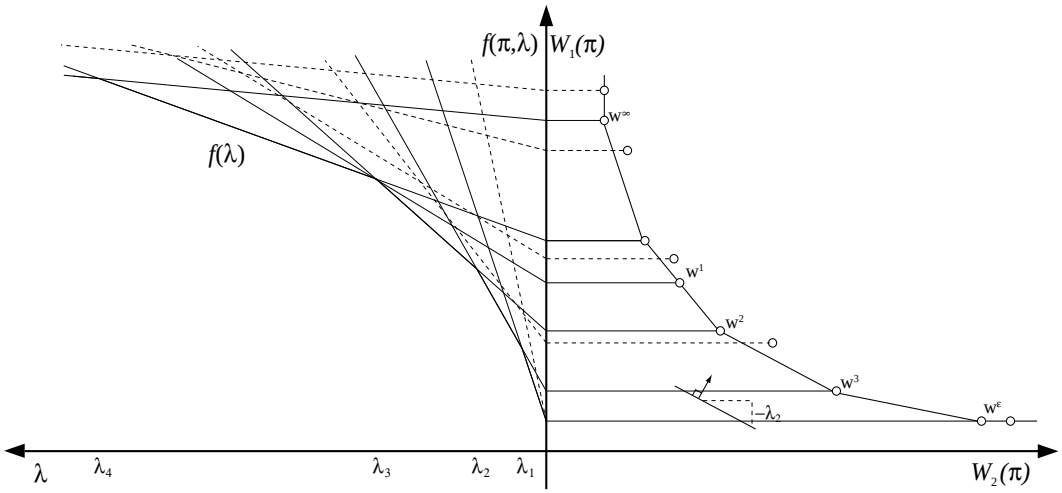



Figure 3.3: The criterion space and its corresponding parametric space.

frontier $\mathcal{W}^{=}$ of the criterion space. This can be seen in Figure 3.3 where all nondominated points inside the triangles correspond to dashed lines which all lie above $f(\lambda)$. The upper/left and lower/right point can be found by solving (3.4) with $\lambda$ very high and $\lambda$ close to zero respectively. Therefore the upper/left and lower/right point are often referred to as $W^{\infty}$ and $W^{\varepsilon}$. ■

Solving Problem (3.4) for digraphs, corresponds to solving a shortest path problem with weight $w_1(e)\lambda + w_2(e)$ on each arc. Therefore a NISE like procedure for digraphs first finds the upper/left and lower/right point. Then calculate the slope of the line between the two points and search in that direction. If a new supported extreme nondominated point is found, we get two new triangles which are searched like before; otherwise we stop. Notice that it is possible to find a supported nonextreme nondominated point $\tilde{W}$ if the search direction is equal to the normal of the line of two adjacent supported extreme nondominated points $\tilde{W}^1$ and $\tilde{W}^2$. Hence $\tilde{W}^1, \tilde{W}^2$ and $\tilde{W}$ all have same minimal parametric weight, and the point found depends on the SBT procedure.
Let *the parametric weighting function* for a hyperpath $\pi$ denote the weighting function

$$W_\lambda(v) = \begin{cases} w_\lambda(e_\pi(v)) + F(e_\pi(v)) & v \in \mathcal{V} \setminus \{s\} \\ 0 & v = s \end{cases} \tag{3.5}$$

where $e_\pi(v)$ denotes the predecessor of node $v$ and $w_\lambda(e) = w_1(e)\lambda + w_2(e)$. Using the parametric weighting function does not always give the right solution to (3.4) when we consider hypergraphs. We look at the following cases: Two value weighting functions and two distance weighting functions.

**Two value weighting functions** We consider two value weighting functions, i.e. given a hyperpath $\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ from $s$ to $t$, the weighting function $W_i$ of $\pi$ is

$$W_i(v) = \begin{cases} w_i(e_\pi(v)) + \sum\limits_{u \in T(e_\pi(v))} a^{e_\pi(v)}(u) W_i(u) & v \in \mathcal{V}_\pi \setminus \{s\} \\ 0 & v = s \end{cases}$$

for $i = 1, 2$. If we want to solve SBT using the parametric weighting function, we have to solve the following recursive equations

$$W_\lambda(v) = \begin{cases} \min\limits_{e \in BS(v)} \left\{ w_\lambda(e) + \sum\limits_{u \in T(e)} a^e(u) W_\lambda(u) \right\} & v \in \mathcal{V} \setminus \{s\} \\ 0 & v = s \end{cases} \tag{3.6}$$

Since finding the shortest hyperpath w.r.t. the value function $W_i$ can be formulated as an LP problem, we have that solving (3.6) corresponds to solving an LP problem with the same constraints, but with an objective function cost for hyperarc $e$ on $w_1(e)\lambda + w_2(e)$ instead of $w_i(e)$. Therefore the minimal hyperpath $\pi$ w.r.t. the parametric weighting function satisfies that $W_\lambda(\pi) = W_1(\pi)\lambda + W_2(\pi)$, i.e. solving (3.6) gives us the solution to (3.4).
A NISE like procedure for two value functions which finds all supported extreme nondominated points, is formulated in Procedure 5. In each step we search in the direction of the normal of the line between $W^+$ and $W^-$. On line 5 we find a supported nondominated point $W(\pi)$. If $f(\pi, \lambda) < W^+_\lambda$ then $W(\pi)$ must be a supported extreme nondominated point and we add it to $\Phi$ on line 6.

**Two distance weighting functions** We here consider two distance functions, i.e. given hyperpath $\pi$

$$W_i(v) = \begin{cases} w_i(e_\pi(v)) + \max\limits_{u \in T(e_\pi(v))} \{W_i(u)\} & v \in \mathcal{V}_\pi \setminus \{s\} \\ 0 & v = s \end{cases}$$

**Precondition:** Let $\left(W^{+}, W^{-}\right)$ define a search direction, and let $\Phi$ be an ordered nondominated set. Given $W \in \Phi$ let $W^{next}$ denote the point following $W$ in $\Phi$, if $W$ is the last element in $\Phi$ then $W^{next} := null$.

**Initialization:** Use an SBT procedure to find the upper/right point $W^{\infty}$ and the lower/left point $W^{\varepsilon}$.

**if** $(W^{\infty} = W^{\varepsilon})$ **then** stop (there is only one nondominated solution)
**else** set $\Phi = \{W^{\varepsilon}, W^{\infty}\}$, $W^{+} = W^{\infty}$, $W^{-} = W^{\varepsilon}$
**while** $(W^{+} \neq W^{\varepsilon})$ **do** set $\lambda = \left|(W_2^{-} - W_2^{+})/(W_1^{-} - W_1^{+})\right|$, $W_{\lambda}^{+} = W_1^{+}\lambda + W_2^{+}$
  **for** $(e \in \mathcal{E})$ **do** set $w_{\lambda}(e) = w_1(e)\,\lambda + w_2(e)$
  find the minimal hyperpath $\pi$ w.r.t. the parametric weighting function
  **if** $(f(\pi, \lambda) < W_{\lambda}^{+})$ **then call** procedure insert$(W(\pi), \Phi)$
    $W^{-} = W(\pi)$
  **end if**
  **else** set $W^{+} = W^{-}$, $W^{-} = W^{+next}$
**end while**

Procedure 5: Finding all supported extreme nondominated points (value/value).

for $i = 1, 2$. If we solve SBT using the parametric weighting function we have to solve the following recursive equations

$$W_{\lambda}(v) = \begin{cases} \min\limits_{e \in BS(v)} \left\{ w_{\lambda}(e) + \max\limits_{u \in T(e)} \{W_{\lambda}(u)\} \right\} & v \in \mathcal{V} \setminus \{s\} \\ 0 & v = s \end{cases} \tag{3.7}$$

Because no LP formulation for the distance exists, finding the shortest hyperpath w.r.t. the parametric weighting function does not always give the same solution to (3.4), as can be seen in the following example.

**Example 3** Consider the hypergraph in Figure 3.4 which contains two hyperpaths and assume that we want to find the shortest hyperpath using the parametric weighting function with $\lambda = 1$. This gives shortest hyperpath $\pi_1$ with predecessor hyperarc $e_4$ in node $v_4$. The weights are $W_{\lambda}(v_4) = 14 \Rightarrow W_{\lambda}(t) = 30$. For the same hyperpath, we have that the weights w.r.t. the first and second weight is $W_1(t) = 12$ and $W_2(t) = 20 \Rightarrow f(\pi_1, \lambda) = 32$, hence,

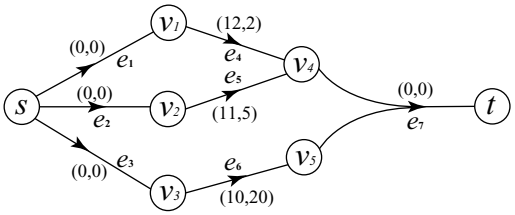



Figure 3.4: A hypergraph where the minimal hyperpath w.r.t. the parametric weighting function don't give the optimal solution of Problem (3.4).

$W_{\lambda}(t) \neq W_1(t)\,\lambda + W_2(t)$. Furthermore, hyperpath $\pi_2$, with $e_5$ as predecessor hyperarc in node $v_4$, gives first and second weight $W_1(t) = 11$ and $W_2(t) = 20 \Rightarrow f(\pi_2, \lambda) = 31$, i.e. $\pi_2$ is the optimal hyperpath to (3.4). ■

Example 3 shows that solving the recursive equations in (3.7) do not always give the right solution to (3.4). However, since the parametric weight of (3.7) is below the actual parametric weight, the parametric weighting function gives us a lower bound (see Manuscript I, Section 4, Theorem 6 for more details).

**Precondition:** Let $\left(W^{+}, W^{-}\right)$ define a search direction and let $\Phi$ be an ordered nondominated set of points. Given $W \in \Phi$ let $W^{next}$ denote the point following $W$ and let $W^{back}$ denote the point before $W$, if $W$ is the last element in $\Phi$ then $W^{next} := null$. Let increase denote a boolean.

**Initialization:** Use an SBT procedure to find the upper/right point $W^{\infty}$ and the lower/left point $W^{\varepsilon}$.

1 **if** $(W^{\infty} = W^{\varepsilon})$ **then** stop (there is only one nondominated solution)
2 **else** set $\Phi = \{W^{\varepsilon}, W^{\infty}\}$, $W^{+} = W^{\infty}$, $W^{-} = W^{\varepsilon}$
3 **while** $(W^{+} \neq W^{\varepsilon})$ **do** let $\lambda = \left|(W_2^{-} - W_2^{+})/(W_1^{-} - W_1^{+})\right|$, increase $= false$
4 **for** $(e \in \mathcal{E})$ **do** set $w_{\lambda}(e) = w_1(e)\,\lambda + w_2(e)$
5 find the minimal hyperpath $\check{\pi}$ w.r.t. the parametric weighting function
6 **call** procedure insert$(W(\check{\pi}), \Phi)$
7 **if** ($W(\check{\pi})$ nondominated and $W_1(\check{\pi}) \leq W_1^{+}$) **then** $W^{+} := W(\check{\pi})^{back}$
8 **if** ($W(\check{\pi})$ dominated or $W_1(\check{\pi}) > W_1^{-}$) **then** increase = *true*
9 find the minimal hyperpath $\hat{\pi}$ w.r.t. the upper bound weighting function (3.8)
10 **if** $(\hat{\pi} \neq \check{\pi})$ **then call** procedure insert$(W(\hat{\pi}), \Phi)$
11 **if** ($W(\hat{\pi})$ nondominated and $W_1(\hat{\pi}) \leq W_1^{+}$) **then** $W^{+} := W(\hat{\pi})^{back}$
12 **if** (increase and ($W(\hat{\pi})$ dominated or $W_1(\hat{\pi}) > W_1^{-}$)) **then** $W^{+} := W^{next}$
13 **end if**
14 set $W^{-} := W^{+next}$
15 **end while**
16 remove from $\Phi$ all nonextreme points

Procedure 6: Finding an approximation of the supported extreme nondominated points (distance/distance).

Notice if $\hat{\pi}$ denotes the shortest hyperpath w.r.t. the parametric weighting function, we have that $W_{\lambda}(\hat{\pi})$ is a lower bound on $f(\lambda)$ but $W(\hat{\pi}) = (W_1(\hat{\pi}), W_2(\hat{\pi}))$ is not necessarily a supported extreme nondominated point.
Using the parametric weighting function gives $W_{\lambda}(\pi) \leq W_1(\pi)\,\lambda + W_2(\pi)$, nevertheless there exists another weighting function satisfying $\tilde{W}(\pi) = W_1(\pi)\,\lambda + W_2(\pi)$

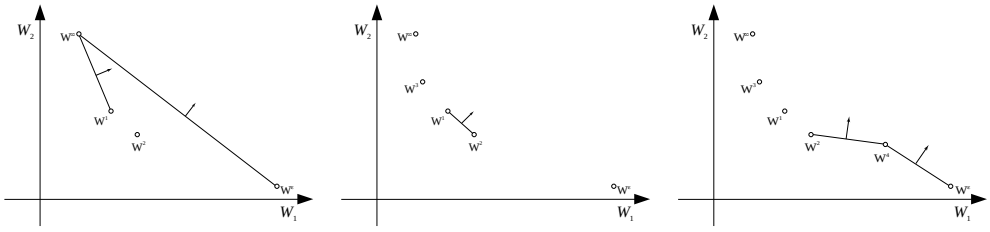
$$\tilde{W}(v) = \begin{cases} 0 & v = s \\ \max\limits_{u \in T(e_{\pi}(v))} \{W_1(v)\}\,\lambda + \max\limits_{u \in T(e_{\pi}(v))} \{W_2(v)\} + w_{\lambda}(e_{\pi}(v)) & v \in \mathcal{V}_{\pi} \backslash \{s\} \end{cases} \tag{3.8}$$

Here we have 3 labels in each node: $W_1$ and $W_2$ which are used to calculate $\tilde{W}$. If we solve SBT with weighting function (3.8), we have to solve the following recursive equations

$$\tilde{W}(v) = \begin{cases} 0 & v = s \\ \min\limits_{e \in BS(v)} \left\{ \max\limits_{u \in T(e)} \{W_1(v)\}\,\lambda + \max\limits_{u \in T(e)} \{W_2(v)\} + w_{\lambda}(e) \right\} & v \in \mathcal{V} \backslash \{s\} \end{cases} \tag{3.9}$$

Solving the recursive equations (3.9) find a minimal hyperpath $\pi$ which often corresponds to a supported extreme nondominated point, but $\pi$ is not necessarily the hyperpath which solves (3.4). This can be seen in Figure 3.4 where hyperpath $\pi_1$ is the solution of equations (3.9). However, $\pi_2$ is the hyperpath which solves (3.4). Therefore $\tilde{W}(\pi)$ is an upper bound on $f(\lambda)$.
Sometimes a solution found using the upper bound weighting function (3.8) cannot be found using the parametric weighting function. Therefore, by combining the two functions, we can find a better approximation of the frontier.
Procedure 6 finds an approximation of the supported extreme nondominated points. The procedure is illustrated by the following example.

**Example 4** Assume that the initialization finds the points $W^{\infty}$ and $W^{\varepsilon}$ in Figure 3.5(a). First iteration now searches in the direction of the normal to the line defined by $W^{\infty}$ and $W^{\varepsilon}$. Suppose that $W^1$ is the point found on line 5. Because $W^1$ is nondominated and



(a) Iteration 1 and 2 (b) Iteration 3 and 4 (c) Iteration 8 and 9

Figure 3.5: Finding supported extreme nondominated points with Procedure 6 on the facing page.

between $W^+$ and $W^-$ the conditions on line 7 and 8 fail. Assuming that line 9 finds the same point, line 14 now sets $W^-$ to $W^1$. Second iteration now searches in the direction of the normal to the line defined by $W^\infty$ and $W^1$. If we assume that line 5 finds point $W^1$, line 8 sets increase to *true* because $W^1$ is not a new point inside the triangle. Moreover, if line 9 finds point $W^2$, line 12 now sets $W^+ = W^1$ because both points found in the iteration are dominated or to the right of $W_1^-$.
Iteration 3 now searches the triangle defined by $W^1$ and $W^2$. If line 5 now finds point $W^3$ in Figure 3.5(b), we have that line 7 set $W^+ = W^\infty$ because a new search direction is defined by the points $W^\infty$ and $W^3$. If we assume that line 9 does not find a new point, iteration 4 now searches in this direction. Assuming that no new points are found during iteration 4-6, iteration 7 now searches in the direction defined by $W^2$ and $W^\varepsilon$ and finds for instance the point $W^4$ in Figure 3.5(c). Therefore iteration 8 and 9 will search in the directions shown in Figure 3.5(c) and if no new points are found stop. Note that we sometimes search in the same direction more than once, but since the SBT procedure is so efficient this is now very costly. Moreover, at the end of Procedure 6, the ordered nondominated set $\Phi$ can contain nonextreme nondominated points. Therefore these are removed on line 16. ■

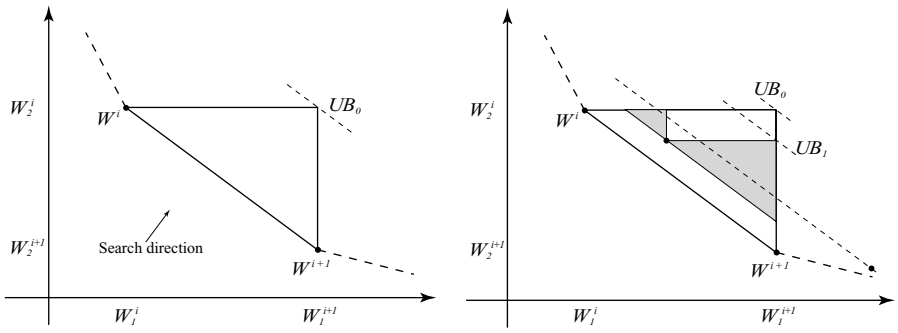
Procedure 6 finds a set of nondominated extreme points. This set is not necessarily equal to the set of supported extreme nondominated points in $\mathcal{W}_{Eff}$. Therefore it is possible that some points $W \in \Phi$ are dominated by a point in $\mathcal{W}_{Eff}$.

**Phase 2: Finding unsupported nondominated solutions**

Assume that the first phase has been completed, i.e. an ordered nondominated set

$$\Phi = \left\{W^1, W^2, ..., W^{q+1}\right\}$$

of supported extreme nondominated points have been found. Let $\Delta = \{\Delta_1, ..., \Delta_q\}$ denote the triangles or gaps defined by $\Phi$ (see Figure 3.6(a)). Second phase searches each triangle using a $k'th$ best procedure until all unsupported nondominated points inside the triangle have been found. This is done by using modified weights $w_\lambda(e)$ on hypergraph $\mathcal{H}$ and hence the $k'th$ best procedure will search in the direction of the normal to the line between the two points which define the triangle (see Figure 3.6(a)). The procedure stops when an upper bound has been reached. At start the upper bound is $UB_0 = W_1^{i+1}\lambda + W_2^i$. However, when a new unsupported nondominated point is found, we calculate an new upper bound (see Figure 3.6(b)). Furthermore, if an interactive approach is used, the decision maker can set an upper bound where he is satisfied. Note that when we use a $k'th$ best procedure it is possible to find points outside the triangle (see Figure 3.6(b)). These points are not checked for dominance, instead the next $k'th$ solution is calculated. How the $k'th$ like procedure performs depends on the weighting functions considered and in which order the solutions




(a) Triangle $\Delta_i$ of $\Delta$ (b) Upper bound of $\Delta_i$.

Figure 3.6: Triangle search.

**Precondition:** Let $\Delta = \{\Delta_1, ..., \Delta_q\}$ denote the triangles of the supported extreme nondominated points and let $\Phi$ be an ordered nondominated set.

**Initialization:** Set $\Phi = \emptyset$

```
1  for (i = 1 to q) do consider triangle Δ_i and
2      set λ = |W_2^i − W_2^{i+1} / W_1^i − W_1^{i+1}|, Φ := Φ ∪ {W^i, W^{i+1}}
3      ub = W_1^{i+1} λ + W_2^i, lb = 0 and k = 1
4      for (e ∈ E) do set w_λ(e) = w_1(e) λ + w_2(e)
5      while (lb < ub) do use a k'th procedure and the parametric weighting function
6          to find the k'th best solution W_λ^k = W_1^k λ + W_2^k of H
7          set lb = W_λ^k
8          if (W_1^k < W_1^{i+1} and W_2^k < W_2^i) then (i.e. we are in the triangle)
9              call procedure insert(W^k, Φ).
10             if (W^k nondominated) then update ub
11         end if
12         k := k + 1
13     end while
14 end for
```

Procedure 7: Finding nondominated points (value/value).

are found. We again consider 2 choices of weighting function: Two value weighting functions and two distance weighting functions.

**Two value weighting functions** Here Procedure 5 on page 17 finds all supported extreme points of $\mathcal{W}_{Eff}$ and therefore, when we start the second phase, we know how the triangles are defined. Moreover, a minimal hyperpath $\pi_\lambda$ which is the solution of (3.4) can be found using the parametric weighting function (3.5). We can now find all unsupported nondominated points using Procedure 7. Line 5 in Procedure 7 finds a new *k'th* hyperpath while $W_\lambda^k$ is below the upper bound. If $W^k$ is a nondominated point inside the triangle, we update the upper bound on line 10. Note when we use procedure insert to insert $W^k$ in $\Phi$ on line 9, we do not have to search $\Phi$ from start because we know $W_1^k > W_1^i$, i.e. we always start to search $\Phi$ from $W^i$ in procedure insert.

**Two distance functions** When two distance weighting functions are considered, first phase (Procedure 6 on page 18) does not necessary find all supported extreme nondominated points of $\mathcal{W}_{Eff}$ but just an approximation of $\mathcal{W}_{Eff}$. This only gives us an approximation of the triangles at the beginning of second phase. Assume that the approximation are the

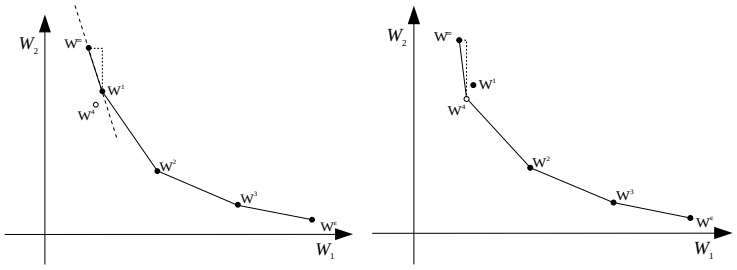
**Precondition:** Let $\Phi = \{W_1, ..., W_q\}$ denote an ordered nondominated set containing the approximation of the supported extreme nondominated points found in phase one. Given $W \in \Phi$, let $W^{next}$ denote the point following $W$ and let $W^{next}$ be equal to $null$ if $W$ is the last point in $\Phi$.

**Initialization:** $W^+ = W^1$

1 **while** $(W^{+next} \neq null)$ **do** $W^- = W^{+next}$, $\lambda = \left|W_2^- - W_2^+ / W_1^- - W_1^+\right|$
2 $ub = W_1^+\lambda + W_2^-$, $lb = 0$, $k = 1$, $W_{frontier} = W_1^+\lambda + W_2^+$, $newf = false$
3 **for** $(e \in \mathcal{E})$ **do** set $w_\lambda(e) = w_1(e)\,\lambda + w_2(e)$
4 **while** $(lb < ub)$ **do** use a *k'th* procedure and the parametric weighting function
5 to find the *k'th* solution $W_\lambda^k$ of $\mathcal{H}$
6 set $lb = W_\lambda^k$, $W_{now} = W_1^k\lambda + W_2^k$
7 **if** $(W_{now} < W_{frontier}$ and $W_1^+ \leq W_1^k)$ **then**
8 remove all nonfrontier points from $W^+$ in $\Phi$
9 set $newf = true$
10 **break**
11 **end if**
12 **if** $(W_1^k < W_1^-$ and $W_2^k < W_2^+)$ **then call** procedure insert$(W^k, \Phi)$.
13 **if** ($W^k$ nondominated) **then** update $ub$
14 **end if**
15 $k := k + 1$
16 **end while**
17 **if** $(newf = false)$ **then** $W^+ = W^-$
18 **end while**

Procedure 8: Finding nondominated points (distance/distance).

black points given in Figure 3.7(a). Now, if during the search of the triangle defined by $W^\infty$ and $W^1$, the point $W^4$ in Figure 3.7(a) is found, then $W^4$ is a new supported extreme nondominated point because the parametric weight of $W^4$ is below the parametric weight of $W^\infty$. I other words $W^4$ is below the dashed line between the points $W^\infty$ and $W^1$ in Figure 3.7(a)




(a) Approximated frontier solutions and a new frontier solution

(b) New approximated frontier

Figure 3.7: Finding a better approximation during second phase.

Therefore a new approximation of the frontier is found and we search the new triangle defined by $W^\infty$ and $W^4$ (see Figure 3.7(b)). Let $\Phi$ be an ordered nondominated set containing all current nondominated points found by first and second phase and let $W^+$ denote the upper/left point of the triangle where the new supported extreme point is found. Then by removing all nonfrontier points above $W^+$ in $\Phi$, we remove a point such as $W^1$ in Figure 3.7(b). Moreover, the new triangle we have to search is defined by $W^+$ and the point following it in $\Phi$. The second-phase procedure for two distance functions is stated in Procedure 8. The boolean $newf$ on line 2 in Procedure 8 is set to true if a new frontier point is found. $W_{frontier}$ denotes the parametric weight of points on the line defined by $W^+$

and $W^-$. Therefore if $W_{now}$, the parametric weight of the current $k'th$ solution, is below $W_{frontier}$ (line 7), a new supported extreme nondominated point has been found. Line 8 now removes all the unsupported points above $W^+$, $newf$ is set to $true$ and we exit the while loop on line 10. Because $newf = true$ we now search for nondominated points in the triangle defined by $W^+$ and the new supported extreme nondominated point found. If no new frontier point is found during the inner while loop, we increase $W^+$ on line 17 and start searching the next triangle.
The bi-SBT problem is much harder to solve when considering two distance functions. This is due to the fact that one nondominated point often corresponds to many different hyperpaths, as a result the $k'th$ procedure used has to search more hyperpaths. Furthermore, since we use the parametric weighting function, only giving us a lower bound, many hyperpaths with an actual parametric weight over the upper bound is found. By using the upper bound function (3.8) instead of the parametric weighting function on line 4, we find hyperpaths with parametric weight equal to the actual parametric weight. However, since the parametric weight is an upper bound, we do not know whether all nondominated points are found when the procedure stops. Therefore using upper bound function (3.8) on line 4 instead of the parametric weighting function, give us an approximation of the nondominated points. Nevertheless, computational testing shows that the approximation is good and much faster to calculate.

## 3.3 Computational results

In Manuscript I other procedures to solve the bi-SBT problem were developed and tested against each other. The following procedures were examined:

***k'th* simple** We here use an extension of the simple search procedure presented in [7]. We here first, find an upper bound on weight one by finding the shortest hyperpath w.r.t. weight two. Next, we use a *k'th* procedure in weight ones direction to find all hyperpaths below this upper bound.

***k'th* diagonal** Instead of searching each triangle, we here use only one *k'th* procedure to search in the direction of the normal to the line between the upper/left and lower/right point.

***k'th* diagonal (lb)** Equal to *k'th* diagonal, but use the *k'th* procedure which branches on lower bounds, that is finds all solutions below an upper bound (Procedure 4).

**Two-phases** We here consider the two-phases procedures in Section 3.2.1 together with a normal *k'th* procedure.

**Two-phases (lb)** Equal to two-phases, but second phase use the *k'th* procedure which branches on lower bounds (Procedure 4).

If acyclic hypergraphs are considered, we use an acyclic SBT procedure. Moreover, both the parametric and the upper bound weighting function are used, if two distance weighting functions are considered.
The procedures were implemented in C++ and tested on 60 randomly generated hypergraphs with the following properties:

1. The hyperarc size is between 3 and 5.
2. The weights for each arc is between 1 and 1000, and for each true hyperarc between 1 and 100. This favor hyperpaths with true hyperarcs.
3. Hypergraphs 1-30 are dense with the number of nodes between 100 and 1000. The average number of hyperarcs in the backward star of a node are between 63 and 105.

4. Hypergraphs 31-60 are sparse with the number of nodes between 1000 and 10000. The average number of hyperarcs in the backward star of a node are between 4.4 and 8.6.

The value function used for testing is the sum function, i.e. where all multipliers are equal one. We can summarize the test results as follows:

1. Branching on lower bounds works well. More than 95% of the lower bounds found where equal to the true minimum weight. As a result, using a lower bound *k'th* procedure to calculate nondominated points improve the CPU time dramatically.

2. The two-phases procedure performs best, except for sparse hypergraphs when considering two sum functions. However, the two-phases procedure is more stable since the diagonal procedures sometimes stop before all nondominated points are found (max 500 hyperpaths picked were allowed).

3. It is possible to find all nondominated solutions in relatively short time when two sum functions are considered.

4. If we consider two distance functions the problem is harder to solve. This is due to a much higher density of points inside the areas the two-phases procedure have to search.

5. Because the parametric weighting function gives a week lower bound in the distance case, using the upper bound function instead gives better results. However, the upper bound function only finds an approximation.

6. First phase finds supported extreme nondominated points fast compared to the second phase. The first phase only finds an approximation in the distance case. However, this approximation is very good.

7. Using the acyclic procedures on acyclic hypergraphs give a high reduction in the CPU time. This is mainly due to the acyclic SBT procedure. Moreover, the fact that branching on lower bounds give us the true minimal weight also contribute to the reduction.

For more details on the results see Manuscript I, section 6.

# Chapter 4

# A parametric analysis of the shortest hyperpath problem

In this chapter we consider my temporary results on the parametric shortest hyperpath problem *(PSBT)* in $B$-hypergraphs. It is assumed that the lengths of the hyperarcs in the $B$-hypergraph depend on some parameter $\lambda$. We present procedures for determining the shortest hyperpaths from a source node $s$ to all other nodes $u$ hyperconnected to $s$ as a function of the parameter $\lambda$. To the best of our knowledge this problem has not been considered earlier. Each procedure are illustrated with an example.

The problem of determining the parametric shortest paths from a particular node $s$ to all other nodes $u$ in directed graphs has been presented in a number of papers. Probably the best known paper is Young, Tarjant, and Orlin [35]. However, the problem presented in that paper is somewhat simpler than the one presented in this paper. One reason is, that it only concerns digraphs, and another reason is, that there are some restrictions in the way, the parameter $\lambda$ is allowed to vary. In this chapter, $\lambda$ is allowed to vary more freely. This makes our problem more complicated, but probably also more oriented towards applications.

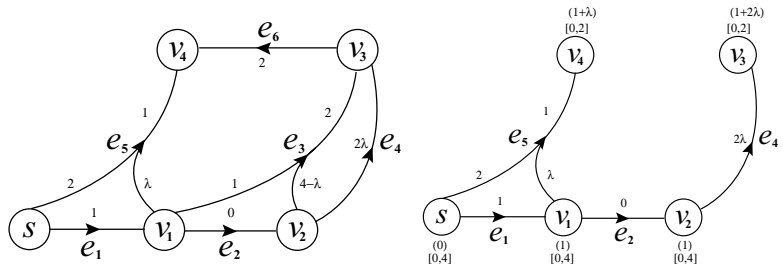
## 4.1 The parametric shortest hyperpath problem

Let $\mathcal{H}=(\mathcal{V},\mathcal{E})$ be a hypergraph where $\mathcal{V}=\{v_1,\ldots,v_n\}$ is the set of nodes and $\mathcal{E}=\{e_1,\ldots,e_m\}$ is the set of hyperarcs. Let $\Lambda=\left[\underline{\lambda},\overline{\lambda}\right]$ and assume that $\lambda\in\Lambda$. Assign to each hyperarc $e\in\mathcal{E}$ a nonnegative weight function $w_e(\lambda)$ and if the value function is considered nonnegative multiplier functions $a_e(u,\lambda)$ for $u\in T(e)$. That is, the weight and the multipliers are functions of the parameter $\lambda\in\Lambda$. We say that the hypergraph $\mathcal{H}$ is parametrized by the parameter $\lambda$ and denote it with $\mathcal{H}_\lambda$. Clearly, for a given value of $\lambda=\lambda_0$, we have that $\mathcal{H}_{\lambda_0}$ is a weighted hypergraph, i.e. weights and multipliers is fixed. Given a hyperpath $\pi$ we let $W_\pi(\lambda)$ denote the parametric weight of $\pi$.

Assume wlog that all nodes $v\in\mathcal{V}$ is hyperconnected to node $s$. In this section we want to determine a shortest hypertree from some source node $s$, i.e. the shortest hyperpaths from node $s$ to all other nodes, as a function of the parameter $\lambda\in\Lambda$. We call this problem the parametric shortest hyperpath problem.

Before presenting the solution procedure we shall illustrate it with a minor example.

**Example 5** A simple parametric hypergraph $\mathcal{H}_\lambda$ is shown in Figure 4.1(a). We consider the value weighting function and the weights and multipliers are written like in Figure 2.2. Now assume that $\Lambda=[0,4]$. The minimal weight hypertree for $\mathcal{H}_0$ is shown in Figure 4.1(b). We now want to find intervals of the parameter $\lambda$ so that the hypertree in Figure 4.1(b) is




(a) Hypergraph $\mathcal{H}_\lambda$ (b) $\mathcal{T}_s$ for $\mathcal{H}_0$

Figure 4.1: A simple parametric weighted hypergraph $\mathcal{H}_\lambda$ with $\Lambda = [0, 4]$ and its minimal hypertree $\mathcal{T}_s$ for $\lambda = 0$.

the optimal (minimal weight) one. By definition a shortest hyperpath from node $s$ to itself has value 0, and is optimal for all $\lambda \in \Lambda$. Also, the subpaths to nodes $v_1$ and $v_2$ are optimal for all values of $\lambda \in \Lambda$ since there is only one hyperpath to nodes 1 and 2. Moreover, the parametric weight of node $v_3$, when $e_4$ is used as a predecessor, is minimal if the parametric weight of node $v_3$, when another hyperarc is used as a predecessor is higher, i.e.

$$1 + 2\lambda \leq 1 + (4 - \lambda) + 2 \Leftrightarrow \lambda \leq 2$$

Hence it follows, that for $\lambda \in [0, 2]$ edge $e_4$ is used as a predecessor in a shortest hyperpath to node $v_3$. Conversely if $\lambda \in [2, 4]$ hyperarc $e_3$ is used as a predecessor in a shortest hyperpath to node $v_3$. For node $v_4$ the result are similar, however, we have to consider the intervals $\lambda \in [0, 2]$ and $\lambda \in [2, 4]$ separately, since the predecessor of node $v_3$ is not the same in the two intervals. If $\lambda \in [0, 2]$ we have

$$2 \cdot 0 + 1 \cdot \lambda + 1 \leq 1 \cdot (1 + 2 \cdot \lambda) + 2 \Leftrightarrow \lambda \geq -2$$

i.e. that for $\lambda \in [0, 2]$ edge $e_5$ is used as a predecessor in a shortest hyperpath to node $v_4$. If $\lambda \in [2, 4]$ we have

$$2 \cdot 0 + 1 \cdot \lambda + 1 \leq 1 \cdot (7 - \lambda) + 2 \Leftrightarrow \lambda \leq 4$$

and hence for $\lambda \in [2, 4]$ edge $e_5$ is used as a predecessor in a shortest hyperpath to node $v_4$. We now have calculated a shortest hyperpath from node $s$ to node $v_i$, for $i = 1, \ldots, 4$ and for all $\lambda \in [0, 4]$, as shown in Table 4.1.

| Node | $\lambda$ | $W_v(\lambda)$ | $p_v(\lambda)$ |
|---|---|---|---|
| 1 | [0,4] | 1 | $e_1$ |
| 2 | [0,4] | 1 | $e_2$ |
| 3 | [0,2] | $1 + 2\lambda$ | $e_4$ |
| | [2,4] | $7 - \lambda$ | $e_3$ |
| 4 | [0,4] | $1 + \lambda$ | $e_5$ |

Table 4.1: The hyperpath for $\lambda \in \Lambda$.

In conclusion the hypertree in Figure 4.1(b) is the minimal weight hypertree for $\lambda \in [0, 2]$. ■

Next let us introduce some notation. Consider a particular node $u \in \mathcal{V}$ hyperconnected to $s$ and let $\pi_{su}$ be a shortest hyperpath from node $s$ to node $u$. We want to determine a set

**Precondition:** Let $\lambda_0$ denote a fixed number in $\Lambda$. Given a hypertree $\mathcal{T}_s$, let $W_v(\lambda)$ denote the weight of node $v$ and let $V_{\mathcal{T}_s} = (v_1, ..., v_n)$ denote a valid ordering of $\mathcal{T}_s$.

**Initialization:** Use an SBT procedure to find the minimal hypertree $\mathcal{T}_s$ of $\mathcal{H}_{\lambda_0}$.

```
1 for (i = 1 to n) do
2     Λ_{v_i} = ⋂_{u∈BN_{v_i}} Λ_u
3     for (e ∈ BS(v_i)) do
4         if (e ≠ p_{v_i}(λ_0)) then
5             Λ_0 = {λ ∈ Λ | W_{v_i}(λ) ≤ w_e(λ) + F_e(λ)}
6             Λ_{v_i} := Λ_{v_i} ∩ Λ_0
7         end if
8     end for
9 end for
```

Procedure 9: Finding values of $\lambda$ for which the minimal hypertree of $\mathcal{H}_{\lambda_0}$ is minimal.

$\Lambda_u$ of values $\lambda$ for which $\pi_{su}$ is a shortest hyperpath to node $u$, i.e.

$$\lambda \in \Lambda_u \Rightarrow \pi_{su} \text{ is a shortest hyperpath for } \mathcal{H}_\lambda \tag{4.1}$$

This problem is denoted *the parametric shortest hyperpath problem* $(PSBT)$[1]. By definition $\Lambda_s = \Lambda$. Furthermore, the weight of node $s$ is always equal to 0, that is

$$W_s(\lambda) = 0 \quad \forall \lambda \in \Lambda_s = \Lambda \tag{4.2}$$

As it turns out, $\Lambda_u$ need not always has to be an interval, but $\Lambda_u$ can be split into disjoint intervals:

$$\Lambda_u \quad = [\underline{\lambda}_u^1, \overline{\lambda}_u^1] \cup [\underline{\lambda}_u^2, \overline{\lambda}_u^2] \cup \ldots \cup [\underline{\lambda}_u^q, \overline{\lambda}_u^q] = \Lambda_u^1 \cup \Lambda_u^2 \cup \ldots \cup \Lambda_u^q$$

Let the *backward node set* $BN_v$ of node $v$ denote the set of nodes $u$ which are in the tail of some arc $e \in BS(v)$, i.e.

$$BN_v = \{u \in \mathcal{V} \,|u \in T(e),\ e \in BS(v)\}$$

Below we consider two cases, namely the general case and the acyclic case.

### 4.1.1 The general case

Consider $\mathcal{H}_\lambda = (\mathcal{V}, \mathcal{E})$ with $\lambda \in \Lambda$ and let $\lambda_0$ be a fixed number in $\Lambda$, we now have the following lemma.

**Lemma 1** Assume that $\mathcal{T}_s$ is the minimal hypertree for $\mathcal{H}_{\lambda_0}$ and that $\pi_{sv}$ is the corresponding shortest hyperpath to node $v$. Let $\Lambda_v$ denote the set

$$\Lambda_v = \Lambda_0 \cap \bigcap_{u \in BN_v} \Lambda_u$$

where

$$\Lambda_0 = \left\{\lambda \in \Lambda \mid w_{p(v)}(\lambda) + F_{p(v)}(\lambda) \le w_e(\lambda) + F_e(\lambda),\ \forall e \in BS(v) \setminus \{p(v)\}\right\}$$

Then $\Lambda_v$ satisfies condition (4.1).

**Proof** Follows immediately. ∎

Procedure 9 finds the minimum weight hypertree $\mathcal{T}_s$ for $\mathcal{H}_{\lambda_0}$, and calculate sets $\Lambda_v$ satisfying condition (4.1) for $v \in \mathcal{V}$. The procedure starts by using an SBT procedure to find a minimal

[1] Parametric shortest B-tree.

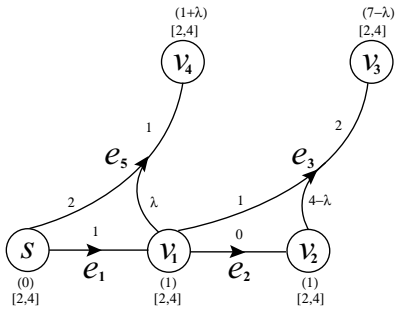



Figure 4.2: The minimal weight hypertree for $\mathcal{H}_2$ with $\Lambda = [2, 4]$.

hypertree $\mathcal{T}_s$ for $\mathcal{H}$. Moreover, if Dijkstra's principle is used, then the order we pick the nodes in procedure SBT define a valid ordering of $\mathcal{T}_s$. Line 2 now finds the union of the intervals of the nodes in $BN_v$ and lines 5 and 6 the set from Lemma 1. Since the number of steps in Procedure 9 are finite, we have that Procedure 9 stops in a finite number of steps, provided that the set $\Lambda_0$ can be found in a finite number of steps.
Procedure 9 can also be used if some part of the minimal hypertree is known in advance. For instance suppose a minimal weight hypertree for the nodes $s, u_1, u_2, ..., u_{k-1}$ is known together with their corresponding sets $\Lambda_i$, satisfying condition (4.1). Then the for-loop in Procedure 9 can be changed from running from 1 to $n$, to run from $k$ to $n$. and only the rest of the minimal hypertree $\mathcal{T}_s$ has to be found in the initialization.
Note that Procedure 9 only finds sets $\Lambda_v$ for which the hypertree of $\mathcal{H}_{\lambda_0}$ is minimal. If we want to find a minimal hypertree for all $\lambda \in \Lambda$, we can use Procedure 9 as a subprocedure to solve this problem. We illustrate this idea by continuing Example 5.

**Example 5** (continued) Procedure 9 calculates the intervals $\Lambda_i = \left[\underline{\lambda}_i, \bar{\lambda}_i\right]$ for $i = 1, ..., 4$ and the minimal weight hypertree for $\mathcal{H}_{\lambda_0}$ (see Figure 4.1(b)). Now define $\Phi$ to be the set of all upper endpoints $(\bar{\lambda}_i)$. In this particular example we have

$$\Phi = \{2, 4\}$$

Assume we pick the minimal number, i.e. 2. We know that the hyperpaths to nodes 1 and 2 in Figure 4.1(b) are minimal for $\lambda = 2$, because 2 is contained in the intervals $\Lambda_1$ and $\Lambda_2$. $\lambda = 2$ is also contained in $\Lambda_3$, however, 2 is an endpoint. This means that there is another hyperarc which gives the same weight of node $v_3$ if $\lambda = 2$ (in this case hyperarc $e_3$). Now if we call Procedure 9 with $\lambda_0 = 2$, $k = 3$ and $\Lambda = [2, 4]$, we get the hypertree and intervals shown in Figure 4.2. We see that the hyperpath $\pi_{sv_4}$ is the same as before so hyperpath $\pi_{sv_4}$ is minimal for $\lambda \in [0, 4]$ . We now could remove 2 from $\Phi$, add all new upper endpoints to $\Phi$, pick the minimal number and repeat the step with $\lambda_0 = 4$. However, since $\lambda_0 = 4 = \bar{\lambda}$ it is not necessary to repeat the step. We have now found a shortest hyperpath from node $s$ to node $v_i$ for $i = 1, ...4$ for all $\lambda \in [0, 4]$ (see Table 4.1).
Note that we have only found one (of possible many) shortest hyperpath for each $\lambda \in [0, 4]$, e.g. if $\lambda = 4$ the hyperpath $\pi_{sv_4}$ in Figure 4.2 is minimal but the hyperpath $\pi_{sv_4}$ with $p_{v_4}(4) = e_6$ instead of $p_{v_4}(4) = e_5$ is also minimal. ■

The procedure, demonstrated in the example above, is stated in Procedure 10 which finds a shortest hyperpath from node $s$ to node $v$ for all $\lambda \in \Lambda$. Here, we add the upper endpoint of $\Lambda_v^1$, for each $v \in \mathcal{V}$, to $\Phi$ on line 2. Note only the endpoint $\bar{\lambda}_v^1$ of the first interval $\Lambda_v^1$ in each node is added to $\Phi$. Next, we pick and remove the minimal endpoint on line 3, and finally, we calculate the new interval used in Procedure 9 on line 5. Note that, when Procedure 9 is called on line 1, the node $v_k$ satisfies that there is more than one hyperarc which can be used

**Precondition:** Assume that each call of Procedure 9 calculates a valid ordering $V_{\mathcal{T}_s} = (v_1, ..., v_n)$ of $\mathcal{T}_s$, weights $W_v(\lambda)$, predecessor hyperarcs and the sets $\Lambda_i$ which can be split into intervals

$$\Lambda_i = \Lambda_i^1 \cup \Lambda_i^2 \cup ... \cup \Lambda_i^{q_i} \quad \text{where } \Lambda_i^j = \left[\underline{\lambda}_i^j, \overline{\lambda}_i^j\right] \quad j = 1, ..., q_i$$

**Initialization:** Set $k = 1$, $\lambda_0 = \underline{\lambda}$ and $\bar{\Lambda} = \Lambda$.

1 **while** $(\lambda_0 \neq \overline{\lambda})$ **do** call Procedure 9 with $\lambda_0$, $\bar{\Lambda}$ and $k$ and set $\lambda_{old} := \lambda_0$
2 $\quad \Phi = \Phi \cup \left\{\overline{\lambda}_i^1\right\}$ for $i = k, ..., n$
3 $\quad$ select and remove $\lambda_0 = \min \Phi$
4 $\quad k := \min\left\{i \mid \exists \overline{\lambda}_i^1 = \lambda_0\right\}$
5 $\quad \Lambda := \Lambda \setminus [\lambda_{old}, \lambda_0[$
6 **end while**

Procedure 10: Finding a minimal hyperpath to all nodes for each $\lambda \in \Lambda$.

as a predecessor for node $v_k$. For instance in Example 5 we have that both hyperpaths $\pi_{sv_3}$ in Figure 4.1(b) and Figure 4.2 are minimal for $\lambda = 2$. However, if we use hyperarc $e_4$ as predecessor the interval $\Lambda_3$ calculated by Procedure 9 becomes $[2, 2]$. As the next example illustrates it is best to pick the predecessor hyperarc which gives the largest interval $\Lambda_v^1$.

**Example 6** Consider the hypergraph in Figure 4.3(a) where $\Lambda = [-2, 2]$ and the value is used as weighting function. When we call Procedure 10, we first run Procedure 9 with $k = 1$, $\Lambda = [-2, 2]$ and $\lambda_0 = -2$. Because there is only one hyperpath to nodes 1 and 2, we have the results stated in Table 4.2. For node 3, hyperarc $e_3$ used as predecessor gives the lowest weight, and we have to solve

$$\begin{aligned}&(2 + \lambda)(2 - \lambda) + 4 \cdot 1 + 1 \leq 4 + 4\\ &\Rightarrow \Lambda_0 = [-\infty, -1] \cup [1, \infty]\\ &\Rightarrow \Lambda_3 = [-2, -1] \cup [1, 2]\end{aligned}$$

At node 4, we have that the weight $W_{v_4}(-2)$ using $e_5$ as predecessor hyperarc is equal to the weight using $e_6$ as predecessor hyperarc, so which hyperarc we pick as predecessor hyperarc depends on which rule we use to pick the predecessor hyperarc in Procedure 9. Assume that we pick $e_5$ first. Then we have to solve

$$\begin{aligned}&2(2 + \lambda) + 6 \leq 9 - \lambda^2 + 1\\ &\Rightarrow \Lambda_0 = [-2, 0]\\ &\Rightarrow \Lambda_4 = [-2, -1]\end{aligned}$$

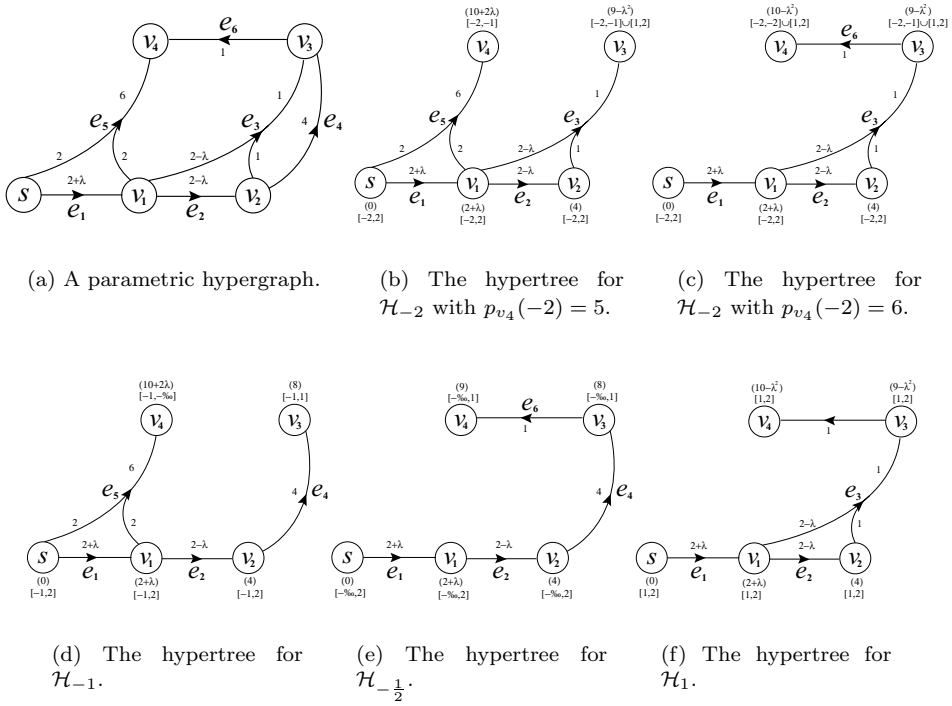
and we get the hypertree shown in Figure 4.3(b). On the other hand, if we first pick $e_6$ we get the opposite

$$\Lambda_4 = [-2, -2] \cup [-1, 2]$$

This hypertree is shown in Figure 4.3(c). Suppose that the hypertree in Figure 4.3(c) was picked. Then an upper endpoint $\overline{\lambda}_4^1 = -2$ where added to $\Phi$ on line 2 and picked again on line 3 making $\Lambda$ on line 5 unchanged. The step is now repeated with $\lambda_0 = -2$, and if we do not have a rule saying that $e_6$ not must be picked as predecessor again, the procedure may

| Node | $\lambda \in$ | $W_v(\lambda)$ | $p_v(\lambda)$ |
|---|---|---|---|
| 1 | $[-2, 2]$ | $2 + \lambda$ | $e_1$ |
| 2 | $[-2, 2]$ | 4 | $e_2$ |

Table 4.2: The predecessor hyperarcs for $\lambda \in [-2, 2]$ when node 1 and 2 are considered.



(a) A parametric hypergraph.

(b) The hypertree for $\mathcal{H}_{-2}$ with $p_{v_4}(-2) = 5$.

(c) The hypertree for $\mathcal{H}_{-2}$ with $p_{v_4}(-2) = 6$.

(d) The hypertree for $\mathcal{H}_{-1}$.

(e) The hypertree for $\mathcal{H}_{-\frac{1}{2}}$.

(f) The hypertree for $\mathcal{H}_1$.

Figure 4.3: A parametric hypergraph and its corresponding minimal hypertrees.

| *node* | $\lambda \in$ | $W_v(\lambda)$ | $p_v(\lambda)$ |
|---|---|---|---|
| 3 | $[-2,-1]$ | $9-\lambda^2$ | $e_3$ |
| | $[-1,1]$ | $8$ | $e_4$ |
| | $[1,2]$ | $9-\lambda^2$ | $e_3$ |
| 4 | $\left[-2,-\frac{1}{2}\right]$ | $10+2\lambda$ | $e_5$ |
| | $\left[-\frac{1}{2},1\right]$ | $9$ | $e_6$ |
| | $[1,2]$ | $10-\lambda^2$ | $e_6$ |

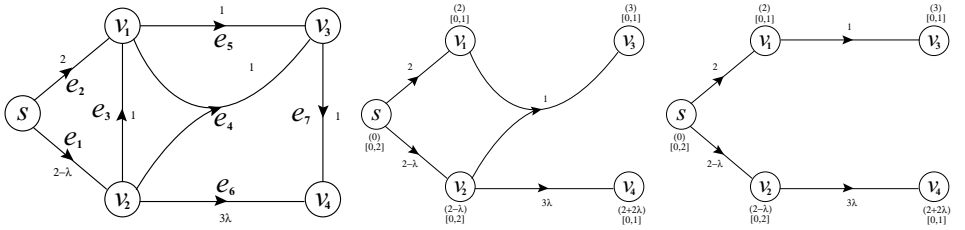
Table 4.3: The predecessor hyperarcs for $\lambda \in [-2,2]$ when node 3 and 4 are considered.

loop. Assume that hyperarc $e_5$ is picked as predecessor, we then get the hypertree in Figure 4.3(b). $\Phi$ now becomes:

$$\Phi = \{-1, 2\}$$

Next Procedure 10 sets $\lambda_{old} = -2$, $\lambda_0 = -1$, $k = 3$ and $\Lambda = [-1, 2]$ and call Procedure 9. Assume that we pick $e_4$ as predecessor hyperarc for $v_3$ instead of $e_3$. The hypertree for $\mathcal{H}_{-1}$ is shown in Figure 4.3(d). Now -$\frac{1}{2}$ is added to $\Phi$ and Procedure 9 is called with $\lambda_0 = -\frac{1}{2}$, $k = 4$ and $\Lambda = \left[-\frac{1}{2}, 2\right]$. The hypertree is shown in Figure 4.3(e). Finally, Procedure 9 is called with $\lambda_0 = 1$, $k = 3$ and $\Lambda = [1, 2]$ which gives the hypertree in Figure 4.3(f). This hypertree is actually identical to the hypertree in Figure 4.3(c) because the intervals in Figure 4.3(f) is a part of the intervals in Figure 4.3(c). We now have a minimal hyperpath $\forall \lambda \in [-2, 2]$ to all nodes in $\mathcal{H}_\lambda$ (see Tables 4.2 and 4.3). ■

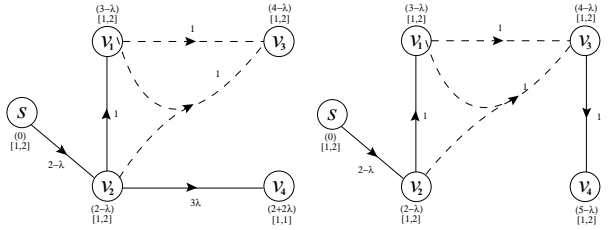
Let $Eq(v)$ be the set of possible predecessor hyperarcs which give minimal weight $W_v(\lambda_0)$ when hypergraph $\mathcal{H}_{\lambda_0}$ is considered. The above example suggests, that a possible rule for finding a minimal hypertree in Procedure 9 is:



(a) A parametric hypergraph.

(b) The hypertree for $\mathcal{H}_0$ with $p_{v_3}(0) = 4$.

(c) The hypertree for $\mathcal{H}_0$ with $p_{v_3}(0) = 5$.



(d) The hypertrees for $\mathcal{H}_1$ with $p_{v_4}(1) = 6$.

(e) The hypertrees for $\mathcal{H}_1$ with $p_{v_4}(1) = 7$.

Figure 4.4: A hypergraph $\mathcal{H}_\lambda$ and its corresponding minimal hypertrees.

Given set $\Lambda$ and hypergraph $\mathcal{H}_{\lambda_0}$, use hyperarc $e \in Eq(v)$ as predecessor for node $v$ in hyperpath $\pi_{sv}$, if no other hyperarc in $Eq(v)$ give a higher interval $\Lambda_v^1$.

By using this rule we would find the hypertrees in Figure 4.3 in 4 iterations.

All procedures described so far finds one (of possible many) minimal hyperpath to a node $v$, they do not find all minimal hyperpaths to a node $v$. This may be a problem when doing sensitivity analysis, since it is very likely that the decision maker want to know if there are alternative hyperpaths that gives the same minimal weight. Then a representative set of minimal hyperpaths can be presented to the decision maker from which one can be chosen. However, all minimal hyperpaths can be found, if we for each endpoint in $\Phi$ and hyperarc $e \in Eq(v)$ calculate the minimal hypertree with $e$ as predecessor in node $v$. This can be illustrated with the following example.

**Example 7** Assume that if there exists more than one hyperarc in $Eq(v)$, then Procedure 9 calculate a minimal hypertree and sets $\Lambda_v$, satisfying condition (4.1) for each $e \in Eq(v)$. Now consider the parametric hypergraph in Figure 4.4(a) with $\Lambda = [0, 2]$. If we consider the distance function and call Procedure 9 with $\Lambda = [0, 2]$ and $\lambda_0 = 0$, we get the hypertrees in Figures 4.4(b) and 4.4(c). At nodes 1 and 2 there is only one predecessor hyperarc which is minimal for $\lambda_0$. At node 3 there are two possible predecessor hyperarcs which are both minimal for $\lambda_0$. Hence Procedure 9 calculate a hypertree for each predecessor which gives the figures in 4.4(b) and 4.4(c). We now have found all minimal hyperpaths for $\mathcal{H}_0$ and the intervals where they are minimal. If we now for each of the hypertrees call Procedure 9 with $k = 1$, $\lambda_0 = 1$ and $\Lambda = [1, 2]$, we get the hypertrees in figure 4.4(d) and 4.4(e). Here two trees are shown in one figure because no matter the choice of predecessor in node $v_3$, the predecessor hyperarc for node 4 is the same. The figures show, for instance, that there are two minimal hyperpaths $\pi_{sv_3}$ which both are minimal for $\lambda \in [0, 2]$. ■

### 4.1.2 The acyclic case

If the hypergraph is acyclic the above procedures can be modified. Let $\mathcal{H}_\lambda=(\mathcal{V},\mathcal{E})$ be an acyclic parametric weighted hypergraph with $\lambda \in \Lambda$. Because $\mathcal{H}_\lambda$ is acyclic a valid ordering $V$ of $\mathcal{H}_\lambda$ exists

$$V = (v_0 = s, v_1, ..., v_n)$$

This valid ordering can be used in Procedure 9. Furthermore, an acyclic SBT procedure can be called which use the valid ordering $V$, and hence becomes much faster since no heap is used to sort the nodes in the candidate set.

# Chapter 5

# Future research topics

In this final chapter, I will outline some ideas for future research. Most of the ideas are loosely described.

In Chapter 4 we considered the temporary results on my project about a parametric analysis of the shortest hyperpath problem. There is a number of problems which should be considered before this project is finished. First, we need a discussion of earlier work on parametric shortest paths. So far we have only made a link to the paper by Young et al. [35]. Second, the complexity for the various procedures have to be analyzed in some more depth and parts of the paper should be explained more carefully. In particular, the explanation concerning, if some part of the hypertree is known in advance. Next, applications may be considered. Here the procedures could be used on the random time-dependent shortest path problem, which can be transformed to a shortest hyperpath problem. If e.g. time and cost on some hyperarcs were not known for certain, a parameter $\lambda$ could be introduced on these hyperarcs. Finally, if we assume that only linear weights on each hyperarc is allowed, the weights in each node becomes linear. As a result the comparison of weights in each node becomes simple. This may be implemented in C++ and computational results carried out.

In Section 3.1 the *k'th* procedure, which branched on lower bounds, found all hyperpaths with weight below an upper bound. Moreover, computational testing reveled that it decreased the CPU time by more than 50% compared to the normal *k'th* procedure. This procedure can also be applied to digraphs. To test how the procedure performs compared to other known *k'th* procedures for digraphs may be an interesting research project.

In Section 3.2, I developed procedures to find efficient hyperpaths when we consider the following cases of weighting function: value/value and distance/distance. Here computational testing showed that finding all nondominated points when two distance weighting functions are considered are difficult because the parametric weighting function gives a week lower bound. If another weighting function could be found with a better lower bound, it would improve the performance of the procedures. Another possibility is to find properties which could prune the search tree of the subhypergraphs better. This could be done by, instead of starting a new $k'th$ search every time a triangle is searched, keeping the search tree and calculate new parametric weights for the new search direction. This might be costly but on the other hand, we only pick each hyperpath once.

As shown in Manuscript I, a hypergraph model for the random time-dependent shortest path problem can be formulated. Therefore it is possible to use the bi-SBT problem to solve the bicriterion random time-dependent shortest hyperpath problem. An interesting research topic might therefore be to use the bi-SBT model as a general framework for transportation of hazardous materials. This would include the development of a bicriterion random time-dependent network generator and tests on the hypergraphs generated with it.

Logic is also an interesting research topic. Dawande and Hooker [11], Hooker [19] introduced logic-based sensitivity analysis which makes it possible to do sensitivity analysis of 0-1 problems. This may be used to generate ranges, where the current solution still is optimal, on more specific integer problems. Furthermore, I am currently a participant in a study group, where we are reading a newly published book [20], about logic based methods for optimization. The book presents interesting new methods where logic can be used in a logic-based modelling framework to solve MP problems. These models can sometimes prune the branch and bound search tree and hence solve the problem faster. This theory could be interesting to use on some specific problems.

# Bibliography


[1] Kim A. Andersen, Lars R. Nielsen, Morten Riis, and Anders J. V. Skriver. The facets of the set packing polytope: A logical interpretation. Working paper 3, Department of Operations Research, University of Aarhus, 2000.

[2] Josè A. Azevedo, Joaquim J. E. R. S. Madeira, and Ernesto Q. V. Martins. A computational improvement for a shortest paths ranking algorithm. *EJOR*, 73:188–191, 1994.

[3] Josè Augusto Azevedo, Maria E. O. S. Costa, Joaquim J. E. R. S. Maderira, and Ernesto Q. V. Martins. An algorithm for the ranking of shortest paths. *EJOR*, 69:97–106, 1993.

[4] J. Brambaugh-Smith and D. Shier. An empirical investigation af some bicriterion shortest path algorithms. *EJOR*, 43:216–224, 1989.

[5] R. Cambini, G Gallo, and M. G. Scutella. Flows om hypergraphs. *Mathematical Programming*, 78:195–217, 1997.

[6] Vijay Chandru and John N. Hooker. *Optimization methods for logical inference*. Wiley Interscience, 1999.

[7] J. C. N. Climaco and E. Q. V. Martins. A bicriterion shortest path algorithm. *EJOR*, 11:399–404, 1982.

[8] J. Cohen. *Multiobjective Programming and Planning*. Academic Press, New York, 1978.

[9] 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 and Operations Research*, 26:789–798, 1999.

[10] John R. Current, Charles S. ReVelle, and Jared L. Cohen. An interactive approach to identify the best compromise solution for two objective shortest path problems. *Computers and operational research*, 17(2):187–198, 1990.

[11] M. Dawande and J. N. Hooker. Inference-based sensitivity analysis for mixed integer/linear programming. To appear in Operations Research., Revised Feb 1998.

[12] 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.

[13] G. Gallo, G. Longo, S. Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied Mathematics*, 42:177–201, 1993.

[14] G. Gallo and S. Pallottino. Hypergraph models and algorithms for the assembly problem. Technical Report 6, Dipartimento di Informatica, March 1992.

[15] G. Gallo and M. G. Scutella. Minimum makespan assembly plans. Technical Report 10, Dipartmento di Informatica - Universita di Pisa, 1998.

[16] M. Garey and D. Johnson. *Computers and Intractability. A Guide of the Theory of NP-Completeness*. W. H. Freeman, 1979.

[17] R. W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[18] Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[19] J. N. Hooker. Inference duality as a basis for sensitivity analysis. *Constraints*, 4: 104–112, 1996.

[20] John Hooker. *Logic-Based Methods for Optimization*. Wiley Interscience, 2000.

[21] Dimiter Ivanchev. Sensitivity analysis of network optimization problems. *Yugoslav Journal of Operations Research*, 5(1):95–109, 1995.

[22] R. G. Jeroslow, K. Martin, R. L. Rardin, and J. Wang. Gainfree Leontief substitution flow problems. *Mathematical Programming*, 57:375–414, 1992.

[23] 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.

[24] Elise D. Miller-Hooks and Hani S. Mahmassani. On the generation af nondominated paths in stochastic, time-varying networks. Technical report, University of Texas at Austin, 1998.

[25] Elise D. Miller-Hooks and Hani S. Mahmassani. Optimal routing of hazardous materials in stochastic, time-varying transportation networks. Technical report, University of Texas at Austin, 1998.

[26] J. Mote, I. Murthy, and D. L. Olson. A parametric approach to solving bicriterion shortest path problems. *EJOR*, 53:81–92, 1991.

[27] Lars Relund Nielsen, Daniele Pretolani, and Kim Allan Andersen. On bicriterion shortest hyperpaths (bi-SBT). Technical report, Department of Operations Research, University of Aarhus, 2001. Available at http://www.imf.au.dk/∼relund/.

[28] Lars Relund Nielsen, Daniele Pretolani, and Kim Allan Andersen. A remark on the definition of a B-hyperpath. Technical report, Department of Operations Research, University of Aarhus, 2001. Available at http://www.imf.au.dk/∼relund/.

[29] Danielle Pretolani. A directed hypergraph model for random time dependent shortest paths. *EJOR*, 123:315–324, sep 2000.

[30] Lars Relund and Kim Allan Andersen. Sensitivity analysis of shortest hyperpath problems. Technical report, Department of Operations Research, University of Aarhus, 2001. Available at http://www.imf.au.dk/∼relund/.

[31] A. J. V. Skriver. A classification af bicriteria shortest path (bsp) algorithms. *Asia-Pacific Journal of Operational Research*, 17:199–212, Sep 2000. Available at http://www.imf.au.dk/ ajs/bisurvey.ps.

[32] Ralph E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. Wiley Interscience. Wiley, 1986.

[33] K. Thulasiraman and M.N.S. Swamy. *Graphs: Theory and Algorithms*. Wiley - Interscience, 1992.

[34] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[35] Neal E. Young, Robert E. Tarjant, and James B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21:205–221, 1991.

# On bicriterion shortest hyperpaths (bi-SBT)

Lars Relund Nielsen* Kim Allan Andersen
Department of Operations Research
University of Aarhus
Ny Munkegade, building 530
DK-8000 Aarhus C
Denmark

Daniele Pretolani
Dipartimento di Matematica e Fisica
Universitá di Camerino
Via Madonna delle Carceri
I-62032 Camerino (MC)
Italy

**Abstract**

The bicriterion shortest path problem has been extensively studied for many years. In addition there has recently been a growing focus on dynamic networks including the random time-dependent shortest path problem which can be transformed to the shortest hyperpath problem [24]. But no attempt has been made to find bicriterion dynamic paths.

This paper aims at solveing the bicriterion shortest hyperpath problem (bi-SBT). The first step implies development of procedures to solve the $k'th$ shortest hyperpath problem in order to pave the way for solving the bi-SBT problem. The paper presents new methods of finding the $k'th$ hyperpath by branching on lower bounds; methods that may also be applied to digraphs. We next develop different bi-SPT procedures which are tested against each other on randomly generated hypergraphs. The results obtained show that bi-SBT can be solved for large hypergraphs.

Keywords: B-hypergraphs, SBT, $k'th$ shortest hyperpath, bi-SBT.

## 1 Introduction

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 minimizing 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 some real-world 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, where 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).

*Corresponding author (e-mail: relund@imf.au.dk)

This problem is called the bicriterion shortest path problem, and is $\mathcal{NP}$-hard because there can be exponentially many different efficient paths [11]. Climaco and Martins [5] solved the problem by first finding an upper bound on one criteria and second use a $k'th$ shortest path procedure to find all efficient solutions below that upper bound. In Mote, Murthy, and Olson [19] a two-phases approach was considered. First phase found the supported extreme nondominated points using an LP-relaxation and second phase searched for unsupported nondominated points using a label correcting approach. An algorithm based only on a label correcting method was presented in Brambaugh-Smith and Shier [3]. More recent an interactive approach which finds only a part of the nondominated solutions has been studied [see 7, 8]. For an overview, see Skriver [25].

Several other extensions of the shortest path problem where arc lengths are stochastic, have been considered. Some authors considered the case in which arc costs are random variables [16–18, 23]. Another case is where arc travel time is both stochastic and time dependent. Here the best route is not necessarily an origin-destination path but a strategy which says that if you leave node $v$ at time $x$ you must travel through arc $e$, Hall [12]. Pretolani [24] showed that the best strategy for a random time-dependent network can be found by defining a time expanded hypergraph $\mathcal{H}$ and then finding the shortest hyperpath in $\mathcal{H}$. For hypergraphs, shortest hyperpaths have been well examined and fast algorithms exist [9, 15, 20, 21]. However, no one has to the authors' knowledge tried to find efficient strategies. In other words, because a hyperpath corresponds to a strategy, we want to find efficient hyperpaths or using the terminology in Pretolani [24]; find the best strategy that both minimizes expected travel time and expected cost.

In this paper we solve the bicriterion shortest hyperpath problem (bi-SBT). The problem is solved using a $k'th$ shortest hyperpath procedure. Finding the $k'th$ shortest path for digraphs has been extensively studied in recent years [2, 30]. We extend the problem to hypergraphs and present new fast methods for finding all shortest hyperpaths below an upper bound; methods that may also be applied to digraphs. Different methods for solving bi-SBT are considered. First, we develop a simple $k'th$ search procedure based on the work in Climaco and Martins [5], i.e. we use the $k'th$ shortest hyperpath procedure on one criteria to find all nondominated points below an upper bound. Second, a two-phases procedure is made. Here first step finds all supported extreme nondominated points using a NISE like procedure, Cohen [6]. In second step we find all unsupported nondominated points by searching each triangle defined by the supported extreme nondominated points using a $k'th$ shortest hyperpath procedure. Finally, a diagonal $k'th$ shortest hyperpath procedure is made which, in contrast to the simple $k'th$ search procedure, searches in a combined direction of the first and second criteria.

The paper is organized as follows. Directed hypergraphs are introduced in Section 2. We here consider B-hypergraphs, referred to as hypergraphs, i.e. hypergraphs where there is only one node in the head of each hyperarc. In Section 3 different procedures to find the $k'th$ shortest hyperpath are developed. Furthermore, a procedure finding all shortest hyperpaths with weight below an upper bound is constructed. The bicriterion shortest hyperpath problem is described in Section 4 and three procedures are developed. Section 5 describes the random hypergraph generator (B-MAKER) which is used to generate test-hypergraphs. In Section 6 computational results are reported. All procedures were implemented in C++ and tested on a UNIX workstation. Finally, applications to random time-dependent shortest paths are considered in Section 7. Appendix A describes the data structures used in the C++ program.

## 2 Directed Hypergraphs

A *directed hypergraph*[1] 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

$$e = (T(e), h(e)) \qquad T(e) \subset \mathcal{V} \quad h(e) \subseteq \mathcal{V} \backslash T(e)$$

where $T(e)$ and $h(e)$ denote the *tail* nodes and the *head* node, respectively.
The *cardinality* of hyperarc $e$ is the sum of the tail and head nodes, i.e.

$$|e| = |T(e)| + |h(e)| = |T(e)| + 1$$

If $|e| = 2$ hyperarc $e$ is called an *arc*. The *size* of $\mathcal{H}$ is the sum of the cardinalities of its hyperarcs:

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

We denote by

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

the *forward star* and the *backward star* of node $u$, respectively. A *path* $P_{st}$ in a hypergraph $\mathcal{H}$ is a sequence of nodes and hyperarcs in $\mathcal{H}$:

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

where, for $i = 1, ...q + 1, v_i \in T(e_i)$ and $v_{i+1} \in h(e_i)$. 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 $t \in T(e_1)$. A path is *cycle-free* if it does not contain any subpath which is a cycle, i.e.

$$v_i \in T(e_j) \Rightarrow j \geq i \qquad 1 \leq i \leq q + 1$$

If $\mathcal{H}$ contains no cycles, it is *acyclic*. $\tilde{\mathcal{H}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ is a *subhypergraph* of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ if $\tilde{\mathcal{H}}$ satisfies $\tilde{\mathcal{V}} \subseteq \mathcal{V}$ 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 $\mathcal{H}$.

**Example 1** A hypergraph $\mathcal{H}$ is shown in Figure 1 on the following page. Here we have

$$|e_1| = |T(e_1)| + 1 = 3$$
$$FS(v_2) = \{e_5, e_6, e_7, e_2\}$$
$$BS(v_5) = \{e_4, e_5\}$$
$$size(\mathcal{H}) = 19$$

A path from $v_4$ to $v_6$ is

$$P_{v_4 v_6} = (v_4, e_1, v_2, e_2, v_3, e_7, v_6)$$

and a path from $v_1$ to $v_4$ is

$$P_{v_1 v_3} = (v_1, e_1, v_2, e_2, v_3, e_3, v_4)$$

path $P_{v_1 v_3}$ is actually a cycle because

$$v_4 \in T(e_1) \nRightarrow 1 \geq 4$$

A subhypergraph $\widetilde{\mathcal{H}}$ of $\mathcal{H}$ is emphasized in Figure 1. ■

[1] We here look at B-hypergraphs i.e hypergraphs where a hyperarc only have one node in its head. More general hypergraphs is presented in Gallo et al. [9].

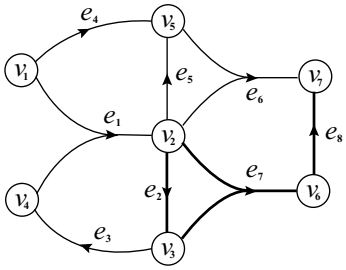



Figure 1: A hypergraph $\mathcal{H}$ with a cycle.

## 2.1 Ordering a hypergraph

We here look at a topological ordering of the nodes of a hypergraph. That is an ordering

$$V = (v_{i_1}, v_{i_2}, ..., v_{i_n})$$

of all the nodes in hypergraph $\mathcal{H}$.

**Definition 1** Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A *valid ordering*

$$V = (v_{i_1}, v_{i_2}, ..., v_{i_n})$$

of the nodes in $\mathcal{H}$. Is a topological ordering of the nodes such that, for each $e \in \mathcal{E}$ and $u \in T(e)$, node $u$ precedes node $h(e)$ in the ordering (see Figure 2).

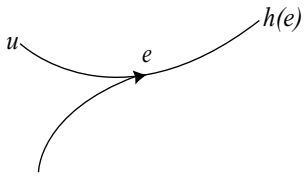



Figure 2: A valid ordering $V = (..., u, ..., h(e), ...)$.

The following theorem is proved in Gallo et al. [9, section 7.2] for $F$-hypergraphs[2], but is also true for B-hypergraphs.

**Theorem 1**

$$\mathcal{H} \text{ acyclic} \Leftrightarrow \text{A valid ordering of the nodes in } \mathcal{H} \text{ is possible}$$

If a valid ordering exists, can it be found with Procedure 1. An acyclic hypergraph $\mathcal{H}$ always has a node $s$ with $BS(s) = \emptyset$, i.e. line 3 first picks a source node. Line 4 adds the node to the valid ordering. The for loop on line 5 decreases $r_i$ until $r_i = 0$, which means that all nodes in $T(e)$ have been added to the valid ordering. If this is not possible, the procedure stops before $k = n$. Because each hyperarc $e$ is examined at most $|T(e)|$ times, the overall complexity is $\mathcal{O}(size(\mathcal{H}))$.

[2] A hypergraph where $|T(e)| = 1$ and $|h(e)| \geq 1$.

**Precondition:** Let $r_i$ denote the number of tailnodes (with repetitions) of the hyperarcs in $BS(i)$ not yet scanned, and let $Q$ be a candidate set implemented as a queue (FIFO). Moreover, $n_u = k$ denotes that node $u$ is the k'th number in the valid ordering $V$.

**Postcondition:** If $k = n$ then $\mathcal{H}$ is acyclic otherwise not.

**Initialization:** Set $k = 0$, $Q = \emptyset$, $r_i = 0\ \forall i \in \mathcal{V}$, $r_i := r_i + |T(e)|\ \forall e = (T(e), \{i\}) \in \mathcal{E}$

```
1 for (i ∈ V) do
2     if (r_i = 0) then Q := Q ∪ {i}
3 while (Q ≠ ∅) do select and remove u ∈ Q
4     set k := k + 1, n_u := k
5     for (e = (T(e), {i}) ∈ FS(u)) do
6         r_i := r_i − 1
7         if (r_i = 0) then Q := Q ∪ {i}
8     end for
9 end while
```

**Procedure 1:** Finding a valid ordering of $\mathcal{H}$.

**Example 2** Consider the hypergraph $\mathcal{H}$ in Figure 3. Here a valid ordering of the nodes in $\mathcal{H}$ is

$$V_1 = (s, v_1, v_2, v_3, v_4, v_5, t)$$

Since there is not a path from $v_4$ to $v_5$, another valid ordering is

$$V_2 = (s, v_1, v_2, v_3, v_5, v_4, t)$$

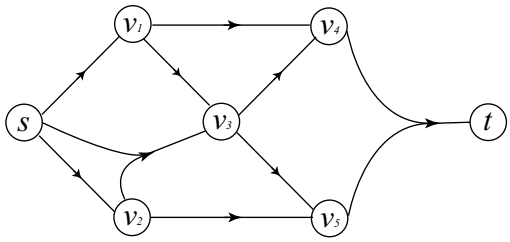



Figure 3: An acyclic hypergraph $\mathcal{H}$.

A valid ordering is therefore not unique. ■

## 2.2 Hyperpaths

We here use a slightly different definition of a hyperpath than in Gallo et al. [9], since in some cases this definition seems to be not working for B-hypergraphs, see Nielsen, Pretolani, and Andersen [22].
Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. A *hyperpath* $\pi_{st}$ of *origin* $s$ and *destination* $t$, is an acyclic minimal hypergraph (with respect to deletion of nodes and hyperarcs) $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ satisfying the following conditions:

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $s, t \in \mathcal{V}_\pi = \cup_{e \in \mathcal{E}_\pi}(T(e) \cup h(e))$
3. $u \in \mathcal{V}_\pi \setminus \{s\} \Rightarrow u$ is connected to $s$ in $\mathcal{H}_\pi$.

**Precondition:** Given $s$, the root of the hypertree, let $p : \mathcal{V} \to \mathcal{E}$ be a predecessor function and $Q$ a candidate set. Let the counter $k_j$ for each hyperarc $e_j$, represent the number of nodes in $T(e_j)$ which have been removed from $Q$, i.e. the head $h(e_j)$ is not examined before all these nodes have been removed.

**Postcondition:** The hypertree is defined by the predecessor function $p$ and if a node $u$ is not hyperconnected to $s$, then $p(u) = 0$.

**Initialization:** Set $Q := \{s\}$, $p(v) := 0\ \forall v \in \mathcal{V}$, $k_j := 0\ \forall e_j \in \mathcal{E}$

```
1 while (Q ≠ ∅) do select and remove u ∈ Q
2     for (e_j ∈ FS(u)) do k_j := k_j + 1
3         if (k_j = |T(e_j)| and p(h(e_j)) = 0) then Q := Q ∪ {h(e_j)}, p(v) := e_j
4     end for
5 end while
```

**Procedure 2:** Visit of a hypergraph $\mathcal{H}$ (B-visit).

Note that 3. implies that for each $u \in \mathcal{V}_\pi \setminus \{s\}$ there exists a hyperarc $e \in \mathcal{E}_\pi$ such that $h(e) = u$, it follows from the minimality that $e$ is unique. Hyperarc $e$ is called the *predecessor* of $u$ and denoted by $e_\pi(u)$. We say that node $t$ is *hyperconnected* to $s$ if there exists a hyperpath $\pi_{st}$. We trivially have

**Corollary 1** Given a hyperpath $\pi_{st}$ and a hyperarc $e \in \mathcal{E}_\pi$, we have that each node $v \in T(e)$ is hyperconnected to $s$.

Hyperpath $\pi_{st}$ is *different* from hyperpath $\pi_{uv}$ if they do not have the same hyperarcs.

**Example 1** (continued) A hyperpath $\pi_{v_2v_7}$ from $v_2$ to $v_7$ is shown in Figure 1 where the hyperpath is emphasized. A hyperpath from $v_1$ to $v_6$ does not exist because the only path from $v_1$ to $v_4$ is a cycle and $v_4$ must be a node in $\mathcal{V}_\pi$, according to Corollary 1. ■

Note that only a subhypergraph of $\mathcal{H}$ has to be considered when we want to find a hyperpath $\pi_{st}$ because the minimality also implies that the following condition holds:

4. $\exists\ u - t\ path \qquad \forall u \in \mathcal{V}_\pi \setminus \{t\}$

Therefore all nodes which do not have a path to $t$ can be removed from $\mathcal{H}$.

## 2.3 Hypertrees

A *directed hypertree* with *root* $s$ is a hypergraph $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_\mathcal{T})$[3] satisfying the following conditions:

1. $\mathcal{T}_s$ is acyclic
2. $\{s\} \cap \mathcal{N} = \emptyset$
3. $BS(s) = \emptyset$
4. $|BS(v)| = 1 \quad \forall v \in \mathcal{N}$

Note a hypertree is the union of hyperpaths to all nodes in $\mathcal{N}$. If $FS(v) = \emptyset$ then $v$ is called a $leaf$. Two hypertrees are shown if Figure 4. Given $\mathcal{H}$, a hypertree can be found with Procedure 2, where the hypertree is defined by a predecessor function $p : \mathcal{V} \to \mathcal{E}$ with $p(v) \in BS(v)$. Note that procedure B-visit finds only one of potentially many hypertrees in $\mathcal{H}$, and it is possible that some hypertrees in $\mathcal{H}$ cannot be found by procedure B-visit. Because each hyperarc is examined at most $|T(e)|$ times the overall complexity is $\mathcal{O}(size(\mathcal{H}))$. Given a hypertree, it is easy to find the hyperpath $\pi_{st}$ to a node $t$ in $\mathcal{N}$

[3] In some definitions it is possible to have more than one root [4].

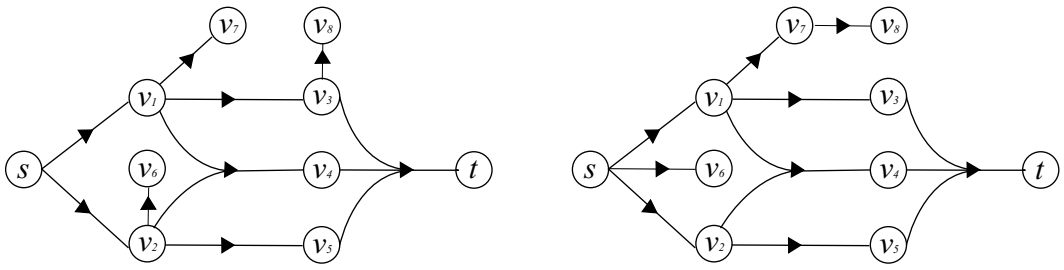



Figure 4: To different hypertrees with the same hyperpath $\pi_{st}$.

**Corollary 2** A hyperpath $\pi_{st}$ in a hypertree $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_{\mathcal{T}})$ is a hypergraph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ which satisfies

1. $\mathcal{V}_\pi = \{\ v \in \mathcal{N} \cup \{s\} \mid \exists path : v - t\}$
2. $\mathcal{E}_\pi = \bigcup_{v \in \mathcal{V}_\pi} BS(v)$

**Proof** Because there is only one hyperarc that points at a node $v$, there is only one unique hyperpath to a node $v$ which is found by backtracking the hypertree from $v$. ■

Note that different hypertrees can have the same hyperpath $\pi_{st}$ as seen in Figure 4.

## 2.4 Weighted hypergraphs

A *weighted hypergraph* is a hypergraph where each hyperarc $e$ is assigned a real weight $w(e)$. Given a hyperpath $\pi_{st}$ a weighting function $W_\pi$ is a node function which assigns weights $W_\pi(u)$ to all nodes in $\pi_{st}$. The weight of hyperpath $\pi_{st}$ is $W_\pi(t)$. We shall restrict ourselves to *additive weighting functions* which is defined by the recursive equations

$$W_\pi(u) = \begin{cases} w(e_\pi(u)) + F(e_\pi(u)) & u \in \mathcal{V}_\pi \setminus \{s\} \\ 0 & u = s \end{cases}$$

where $F(e)$ is a nondecreasing function of the weights of the nodes in $T(e)$. Two particular weighting functions, namely the *distance* and the *value,* have been studied in detail by Gallo et al. [9], and Pretolani [24] who showed that these two functions have practical applications. The *distance* is obtained by defining $F(e)$ as follows:

$$F(e) = \max_{v \in T(e)} \{W_\pi(v)\}$$

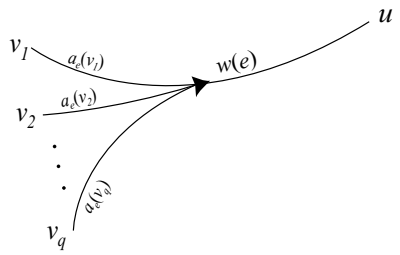



Figure 5: The weights and multipliers of the value function $e = (\{v_1, ..., v_q\}, \{u\})$.

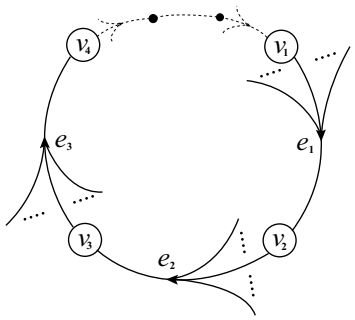



Figure 6: A nondecreasing cycle if condition (2.1) holds.

and the *value* is obtained as follows:

$$F(e) = \sum_{v \in T(e)} a_e(v) W_\pi(v)$$

where $a_e(v)$ is a nonnegative multiplier defined for each hyperarc $e$ and node $v \in T(e)$ (see Figure 5). The distance (the value) of a hyperpath $\pi_{st}$ is the weight of the hyperpath $\pi_{st}$ with respect to the distance (the value) weighting function.

## 2.5 The shortest hyperpath problem

The *shortest hyperpath problem* (*SBT*) [4] consists in finding the minimum weight hyperpaths from an origin $s$ to all nodes in $\mathcal{H}$ hyperconnected to $s$. In general the problem is hard to solve but if the weighting function is additive, fast algorithms exist. We first define a nondecreasing cycle which ensures that no weight can be decreased through a cycle.

**Definition 2** A *nondecreasing cycle* is a cycle $C = \{v_1, e_1, v_2, e_2, ..., v_r, e_r, v_1\}$ that satisfies

$$w(e_r) + F_{v_r}\left[w(e_{r-1}) + F_{v_{r-1}}(...F_{v_2}[w(e_1) + F_{v_1}(z)])\right] \geq z \qquad \forall z \in \mathbf{R}_+ \tag{2.1}$$

Here $F_{v_i}(W)$ denotes the function where $v_i \in T(e_i)$ has weight $W$ and all other nodes $u \in T(e_i)$ has weight equal to zero.

That is, if node $v_1$ has temporary weight $z$ then going through $C$ will give no better temporary weight, see Figure 6. If we consider the distance function, then $F(e) = \max_{v \in T(e)} \{W_\pi(v)\}$ and therefore $F_{v_i}(W) = W$, so condition (2.1) becomes

$$\begin{array}{l} \sum_{i=1}^{r} w(e_i) + z \geq z \\ \Updownarrow \\ \sum_{i=1}^{r} w(e_i) \geq 0 \end{array}$$

which is the normal nonnegativity condition from standard digraphs.
For the value function we have that $F_{v_i}(W) = a_{e_i}(v_i)W$ and condition 2.1 becomes

$$\begin{aligned} & w(e_r) + a_{e_r}(v_r)\left[w(e_{r-1}) + a_{e_{r-1}}(v_{r-1})(...a_{e_2}(v_2)[w(e_1) + a_{e_1}(v_1)z])\right] \\ & = w(e_r) + \sum_{i=1}^{r-1}\left[w(e_{r-i})\prod_{j=0}^{i-1} a_{e_{r-j}}(v_{r-j})\right] + z\prod_{i=1}^{r} a_{e_i}(v_i) \geq z \end{aligned}$$

[4] Shortest B-hypertree.

**Precondition:** Given hypergraph $\mathcal{H}$ with nondecreasing cycles and nonnegative hyperarc weights, let $W(v_i)$ denote the minimal weight in node $v_i$, $F(e)$ the chosen additive weighting function, $Q$ the candidate set and let $p$ be a predecessor function. Moreover, let the counter $k_j$ for each hyperarc $e_j$ represent the number of nodes in $T(e_j)$ which have been removed from $Q$. Therefore we just update $h(e_j)$ when the weight in all nodes in $T(e_j)$ has been calculated.

**Initialization:** Set $W(v_i) = \infty \ \forall i \in \mathcal{V}$, $k_j = 0 \ \forall e \in \mathcal{E}$, $Q = \{s\}$ and $W(s) = 0$

```
1  while (Q ≠ ∅) do
2      select and remove u ∈ Q;
3      for (e_j ∈ FS(u)) do k_j := k_j + 1
4          if (k_j = |T(e_j)|) then v := h(e_j)
5              if (W(v) > w(e_j) + F(e_j)) then
6                  if (v ∉ Q) then
7                      Q := Q ∪ {v}
8                      if (W(v) < ∞) then
9                          for (e_h ∈ FS(v)) do  k_h := k_h − 1
10                     end if
11                 end if
12                 W(v) := w(e_j) + F(e_j), p(v) := e_j
13             end if
14         end if
15     end for
16 end while
```

**Procedure 3:** Shortest B-tree procedure (SBT)

Assuming that $w(e_j)$ is nonnegative, this inequality is obvious true for $z \in \mathbf{R}_+$ if

$$\prod_{i=1}^{r} a_{e_i}(v_i) \geq 1 \tag{2.2}$$

which is the gainfree condition from Jeroslow, Martin, Rardin, and Wang [13].
Now, assume that the weighting function is additive, the weights nonnegative and that all cycles are nondecreasing. Gallo et al. [9] showed that finding the minimum weight hypertree is equivalent to finding a solution to *Bellmans generalized equations*

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

Procedure 3 proposed in Gallo et al. [9] finds the minimum weight hypertree. The main loop starts on line 1. Here a node $u$ is picked and removed from the candidate set on line 2, and line 3 increases $k_j$ by one because node $u \in T(e_j)$ has been removed from $Q$. If all nodes in $T(e_j)$ have been removed from $Q$, i.e. if all weights in $T(e_j)$ have been calculated, we calculate a new weight in $v = h(e_j)$ and check whether the new calculated weight is lower than the old calculated weight on line 5. If so, we have two cases: If $v \in Q$, we update the weight and predecessor function on line 12. If $v \notin Q$, we add $v$ to $Q$ and therefore decrease $k_h$ by one for all $e_h \in FS(v)$ on line 9, provided that $k_h$ is not already zero, and then update the weight and predecessor function on line 12.
The complexity of the procedure depends on the implementation of the candidate set $Q$. If we use Dijkstra's principle, i.e. select from $Q$ a node $u$ satisfying $W(u) = \min\{W(x) \mid x \in Q\}$ at each iteration, then the well-known assumption of nonnegative arc weights in standard digraphs becomes

$$w(e) + F(T(e)) \geq W(x) \qquad \forall x \in T(e), \ e \in \mathcal{E} \tag{2.3}$$

that is, the weight in the head must be equal to or larger than the weight in all the nodes in the tail. If assumption (2.3) is satisfied, then Dijkstra's theorem can be extended to hypergraphs.

**Precondition:** Given $V$, a valid ordering of $\mathcal{H}$. let $p$ denote the predecessor function and $F(e)$ the chosen additive weighting function.

**Initialization:** Set $W(s) := 0$, $W(v_i) = \infty$ $i = 1, ..., n$

```
1 for (i = 1 to n) do
2     for (e ∈ BS(v_i)) do
3         if (W(v_i) > w(e) + F(e)) then W(v_i) := w(e) + F(e), p(v_i) := e
4     end for
5 end for
```

**Procedure 4:** Shortest B-tree procedure when $\mathcal{H}$ is acyclic (SBT-acyclic)

**Theorem 2** Suppose assumption (2.3) is satisfied and $W(u) = \min\{W(x) \mid x \in Q\}$. 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 and therefore line 9 can be dropped. The complexity of sorting $Q$ is $\mathcal{O}(n^2)$ and since each hyperarc $e$ is examined at most $|T(e)|$ times, the overall complexity for scanning all nodes is $\mathcal{O}(size(\mathcal{H}))$. The complexity under Dijkstra's principle now becomes $\mathcal{O}\left(max\left\{n^2, size(\mathcal{H})\right\}\right)$. If a heap implementation is used, the complexity becomes $\mathcal{O}(m\ log\ n)$.
If we have calculated the minimal hypertree for $\mathcal{H}$, and $\mathcal{H}$ is changed to $\tilde{\mathcal{H}}$ by removing some hyperarcs, then the minimal hypertree of $\tilde{\mathcal{H}}$ does not have to be calculated from scratch.

**Theorem 3** Let $\mathcal{T}_s$ be a minimal hypertree with root $s$ of hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ defined by predecessor function $p : \mathcal{V} \to \mathcal{E}$ and let $V$ be a valid ordering of $\mathcal{T}_s$.

$$V = (v_{i_1} = s, v_{i_2}, ..., v_{i_n})$$

Let $\tilde{\mathcal{H}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ be the subhypergraph of $\mathcal{H}$ where some hyperarcs have been removed from

$$BS\left(v_{i_q}\right), ..., BS\left(v_{i_n}\right)$$

then the hypertree for node $v_{i_1}, ..., v_{i_{q-1}}$ defined by

$$p\left(v_{i_j}\right) \quad j = 1, ..., q-1 \tag{2.4}$$

is minimal for $\tilde{\mathcal{H}}$.

**Proof** The hypergraph defined by the predecessor function in (2.4) satisfies all conditions in Section 2.3 and is therefore a hypertree. In addition it is a subhypergraph of $\tilde{\mathcal{H}}$ and since no hyperarcs in $BS(v_i)$, $i = 1, ...q-1$ have been changed in $\tilde{\mathcal{H}}$, it is minimal. ■

Theorem 3 shows that if we apply changes to node $v_{i_q}, ..., v_{i_n}$ then we do not have to calculate the minimal hypertree for node $v_{i_1}, ..., v_{i_{q-1}}$ again.

### 2.5.1 Acyclic case

If $\mathcal{H}$ is acyclic, a fast procedure exists [10]. The procedure is stated in Procedure 4 and needs a valid ordering which can be found with Procedure 1 on page 5.

$$V = (v_0 = s, v_1, ..., v_n)$$

Procedure 4 finds the minimal weight for a node $v$ in one iteration because the valid ordering assures that minimal weights in the predecessor nodes have been found. Again, here each hyperarc $e$ is examined at most $|T(e)|$ times and the overall complexity for scanning all nodes becomes $\mathcal{O}(size(\mathcal{H}))$.

## 2.6 The LP formulation of the value function

As is the case for the shortest path problem, there exists an LP formulation of the shortest hyperpath problem when the value function is considered (this is not true for the distance function). For the value function we have the following problem

$$(P) \quad \begin{array}{ll} \min & \sum_{e\in\mathcal{E}} w_e x_e \\ st. & \sum_{e\in BS(v)} x_e - \sum_{e\in FS(v)} a_e(v)\, x_e = \begin{cases} -1 & v = s \\ 0 & v \in \mathcal{V}\backslash\{s,t\} \\ 1 & v = t \end{cases} \\ & x_e \geq 0 \quad \forall e \in \mathcal{E} \end{array} \tag{2.5}$$

where $x_e$ is the flow through hyperarc $e$. The dual formulation now becomes

$$(D) \quad \begin{array}{ll} \max & q_t \\ st. & w_e + \sum_{v\in T(e)} a_e(v)\, q_v \geq q_{h(e)} \quad \forall e \in \mathcal{E} \\ & q_s = 0 \end{array}$$

Here we actually assume that the primal problem has a flow variable $x_{e_s}$ with a flow of one into node $s$, i.e. equation (2.5) is

$$\sum_{e\in BS(v)} x_e - \sum_{e\in FS(v)} a_e(v)\, x_e = \begin{cases} 0 & v \in \mathcal{V}\backslash\{t\} \\ 1 & v = t \end{cases}$$
$$x_{e_s} = 1 \qquad e_s = (\emptyset, s)$$

So, the primal problem is actually a gainfree Leontief substitution problem [see 13]. The primal solution is not necessarily integral if the minimal hyperpath from $s$ to $t$ contains hyperarcs. But all positive variables in the primal solution define a hyperpath from $s$ to $t$.

**Theorem 4 (Jeroslow et al. [13])** Let $\mathcal{H}$ be a hypergraph fulfilling the gainfree condition (2.2) and let $x_e$ for $e \in \mathcal{E}$ be a solution of $(P)$, then

$$E = \{e \in \mathcal{E} \mid x_e > 0\}$$

defines a minimal hyperpath from $s$ to $t$.

# 3 Finding the *k'th* shortest hyperpath

In this section we describe how to find the $k$-shortest hyperpaths from a root node $s$ to a given destination node $t$ in a hypergraph $\mathcal{H}$, i.e. the shortest hyperpath, the $2'nd$ shortest hyperpath,...,the $k'th$ shortest hyperpath. The procedure is an extension of finding the $k$ shortest loopless paths presented in Yen [30]. Lawler [14] presented similar results in a more general framework where he finds the $k'th$ best solution to a discrete optimization problem. Furthermore, we present a procedure which finds all hyperpaths with a weight below an upper bound. Assume that $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is given and let $\tilde{V}$ be an ordering of the nodes of $\mathcal{H}$.

$$\tilde{V} = (s, u_1, ..., u_n)$$

Because no hyperpath from $s$ to $t$ can contain a hyperarc from $BS(s)$, we assume wlog that $BS(s)$ is empty. Let the hyperarcs be ordered in the following way

$$(\underbrace{e_1^1, e_1^2, ..., e_1^{a_1}}_{\in BS(u_1)}, \quad \underbrace{e_2^1, e_2^2, ..., e_2^{a_2}}_{\in BS(u_2)} \quad , ..., \quad \underbrace{e_n^1, e_n^2, ..., e_n^{a_n}}_{\in BS(u_n)}) \tag{3.1}$$

| | |
|---|---|
| **Step 1** | Use an SBT procedure to find a minimal hyperpath $\pi \in \Pi$. |
| **Step 2** | Divide $\mathcal{H}$ into subhypergraphs $\mathcal{H}_1, ..., \mathcal{H}_q$ which together satisfy: All possible hyperpaths from $s$ to $t$ in $\mathcal{H}$ will be contained in the subhypergraphs except the minimal one found in Step 1. |
| **Step 3** | Use and SBT procedure to calculate a minimal hyperpath $\pi_i$ for every subhypergraph $\mathcal{H}_i$. |
| **Step 4** | Pick a hyperpath $\pi_i$ with minimal weight, of the ones calculated in Step 3. |

**Procedure 5:** Finding the $2'nd$ shortest hyperpath (main steps)

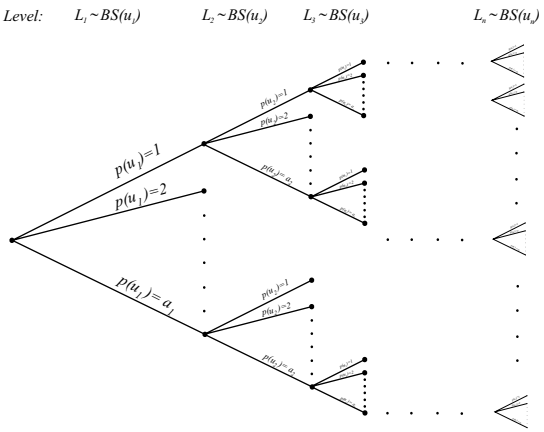



Figure 7: The branching tree $T$ with $n$ levels.

and let the predecessor function $p : \mathcal{V} \to \mathcal{E}$ denote the hyperarc $e \in BS(v)$ used as predecessor hyperarc, i.e. if hyperarc $e_i^l$ is used as predecessor for node $u_i$, then

$$p(i) = l \qquad l \in \{1, ..., a_i\}$$

In the following we let $\Pi$ denote the set of hyperpaths $\pi_{st}$ from $s$ to $t$, i.e.

$$\Pi = \{\pi \subseteq \mathcal{H} \,|\pi \text{ has root } s \text{ and destination } t\}$$

The idea of finding the $2'nd$ shortest hyperpath in $\mathcal{H}$ is now as stated in Procedure 5. But how do we find subhypergraphs that satisfy the condition in step 2? The number of different predecessor functions can be viewed as a branching tree $T$ with $n$ levels as shown in Figure 7 where

1. Each level $L_i$ of branchingarcs corresponds to the possible choices of predecessor hyperarcs of node $u_i$.
2. The root node $t_r$ corresponds to hypergraph $\mathcal{H}$.
3. A leafnode $t_l$ defines a predecessor function or equivalently the subhypergraph obtained from $\mathcal{H}$ by fixing the backward star of node $u_1, ..., u_n$ to the values defined in the path from $t_r$ to $t_l$.
4. Similarly, a treenode $t$ on level $L_i$ corresponds to the subhypergraph obtained from $\mathcal{H}$ by fixing the backward star of node $u_1, ..., u_i$ to the values defined in the path from $t_r$ to $t$.

The minimal hypertree of $\mathcal{H}$ with root $s$ can now be viewed as a path $P_{t_r t_l}$ in $T$ where some of the branchingarcs in $P_{t_r t_l}$ define the minimal hyperpath $\pi$ from $s$ to $t$.

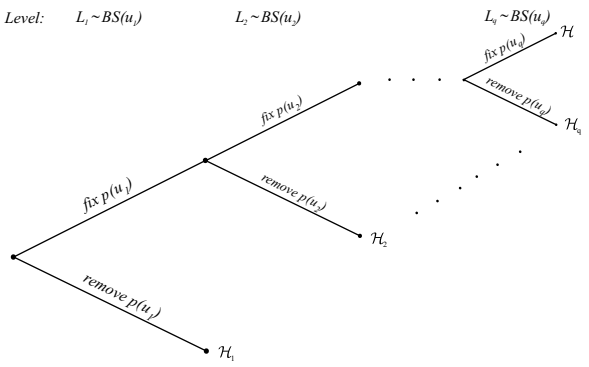



Figure 8: The branching tree for the subhypergraphs.

**Definition 3** Let predecessor function $p$ define the minimal hypertree of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and let an ordering of a subset of nodes $\{u_1, ..., u_l\} \subseteq \mathcal{V}$ be given by

$$V = (u_1, ..., u_l)$$

We say that we *branch on node* $u_i$ meaning that we fix the backward star of node $u_j$ to $p(u_j)$ $(1 \leq j < i)$ and remove $p(u_i)$ from the backward star of node $u_i$. More precisely branching on node $u_i$ corresponds to creating a subhypergraph $\tilde{\mathcal{H}} = \left(\tilde{\mathcal{V}}, \tilde{\mathcal{E}}\right)$ with $\tilde{\mathcal{V}} = \mathcal{V}$ and where the hyperarc set $\tilde{\mathcal{E}}$ is modified in the following way

$$\widetilde{BS}(u_1) = p(u_1), ..., \widetilde{BS}(u_{i-1}) = p(u_{i-1}), \quad \widetilde{BS}(u_i) = BS(u_i) \setminus p(u_i)$$

The ordering $V$ is called *the ordered branching set of* $\mathcal{H}$.

We want to find subhypergraphs that do not contain $\pi$, that is a subtree of $T$ which does not contain the branchingarcs that define $\pi$. A well-known property of hypergraphs is

**Property 1** Let $\mathcal{T}_s$ be a minimal hypertree of $\mathcal{H}$ and $v$ a node which is not in the minimal hyperpath $\pi$ contained in $\mathcal{T}_s$. Then $\pi$ is still a minimal hyperpath for every subhypergraph obtained from $\mathcal{H}$ by deleting or fixing some arcs in $BS(v)$.

Therefore, deleting or fixing some arcs on levels that correspond to nodes that are not in $\pi$, will not change the solution of $\pi$, and hence these levels can be removed from the branching tree. We can now construct $q$ subhypergraphs $\mathcal{H}_1, ..., \mathcal{H}_q$ which satisfy step 2 in Procedure 5 by using the following branching rule

**Branching Rule 1** Let predecessor function $p$ define the minimal hypertree of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and let an ordering of the nodes in the minimal hyperpath $\pi$, without root $s$, be given by

$$V = (u_1, ..., u_q)$$

We now create subhypergraphs $\mathcal{H}_i$ for $1 \leq i \leq q$ by branching on node $u_i$.

The subhypergraphs $\mathcal{H}_1, ..., \mathcal{H}_q$ in Branching Rule 1 correspond to the branching tree $\tilde{T}$ in Figure 8. We here have that all possible choices of predecessor functions, except the minimal one, are contained in $\mathcal{H}_1, ..., \mathcal{H}_q$ and therefore $\mathcal{H}_1, ..., \mathcal{H}_q$ satisfies the conditions in step 2. We now can use Branching Rule 1 in step 2, and hence step 3 and 4 will find the $2'nd$ shortest hyperpath.
Finding the $k'th$ shortest hyperpath works in the same way. Now simply store all the subhypergraphs and their corresponding minimal hyperpath so that it is possible to calculate

**Precondition:** Let $List$ be a candidate set, i.e. $List$ contains all the subhypergraphs, where the $k'th$ shortest hyperpath can be found from, and let $Out$ denote an outputlist. Moreover, let $q$ denote the number of elements in the ordered branching set of $\mathcal{H}^i$.

**Postcondition:** The outputlist $Out$ has been filled with the $k$ shortest hyperpaths.

**Initialization:** Use an SBT procedure to calculate a minimal hyperpath $\pi \in \Pi$ and set $List = \{(\mathcal{H}, \pi)\}$

1 **for** $(i = 1$ to $k)$ **do** pick $\left(\pi^i, \mathcal{H}^i\right) = \arg\min_{\left(\tilde{\pi}, \tilde{\mathcal{H}}\right) \in List} \{W(\tilde{\pi})\}$ and set $Out := Out \cup \left\{\pi^i\right\}$
2 **for** $(j = 1$ to $q)$ **do** create subhypergraph $\mathcal{H}_j$ using Branching Rule 1.
3 calculate $\pi_j = \min_{\pi \in \Pi_{\mathcal{H}_j}} \{W(\pi)\}$ using an SBT procedure.
4 set $List := List \cup \{(\pi_j, \mathcal{H}_j)\}$
5 **end for**
6 **end for**

**Procedure 6:** Finding the $k$ shortest hyperpaths.

new shortest hyperpaths if this minimal hyperpath is used. The $k$ shortest hyperpaths can be found with Procedure 6 which can be illustrated with the following example.

**Example 3** Suppose that we want to find the 4 shortest hyperpaths of hypergraph $\mathcal{H}$ shown in Figure 9. Here its minimal hypertree is emphasized and the weight of the minimal hyperpath is written inside the square. The weighting function considered is the sum function, i.e. the value function with all multipliers equal to one. An ordered branching set of the nodes in the minimal hyperpath $\pi$ is

$$V = (t, 6, 5, 1)$$

The branching tree $\tilde{T}$ corresponding to this branching set is shown in Figure 10 with its corresponding subhypergraphs. Here subhypergraph $\mathcal{H}_4$ is not shown because no hyperpath from $s$ to $t$ exists in $\mathcal{H}_4$. The minimal hypertree of each subhypergraph is emphasized in Figure 11. Now for each subhypergraph $\mathcal{H}_i$ we calculate a minimal hyperpath $\pi_i$ and add $(\mathcal{H}_i, \pi_i)$ to $List$ if the hyperpath exists. We now have

$$List = \{(\mathcal{H}_1, \pi_1), (\mathcal{H}_2, \pi_2), (\mathcal{H}_3, \pi_3)\}$$

We then select and remove an element from $List$ where the hyperpath has minimal weight, i.e. $(\mathcal{H}_1, \pi_1)$ with $W(t) = 14$. Here an ordering of the levels in the branching tree is

$$V = (t, 8, 2)$$

and we have to solve 3 subproblems $\mathcal{H}_{11}$, $\mathcal{H}_{12}$ and $\mathcal{H}_{13}$. Like before here subhypergraph $\mathcal{H}_{13}$ does not have any hyperpaths from $s$ to $t$. The subhypergraphs with finite weight $W(t)$ and

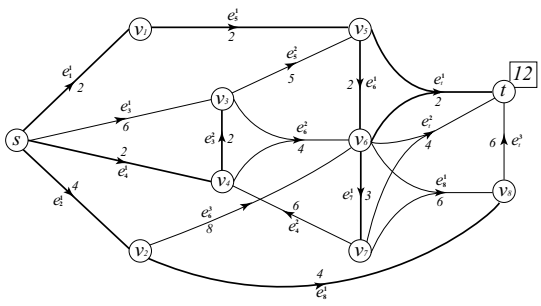



Figure 9: Hypergraph $\mathcal{H}$ with its minimal hypertree emphasized.

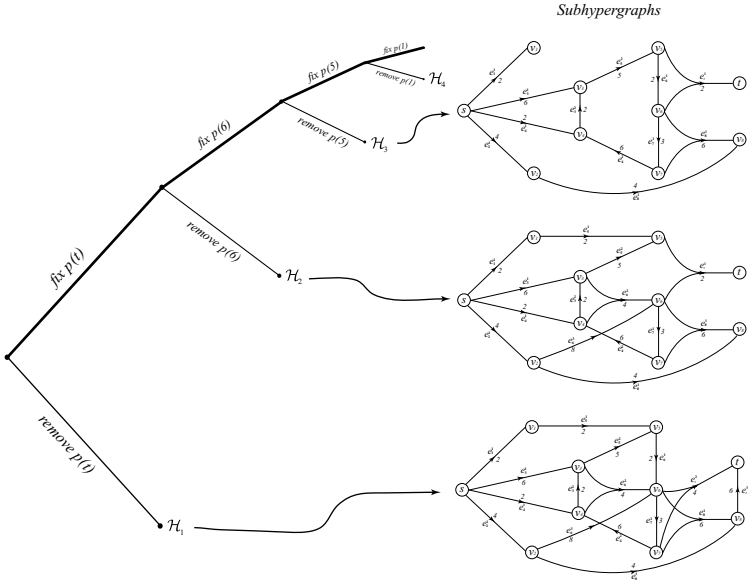
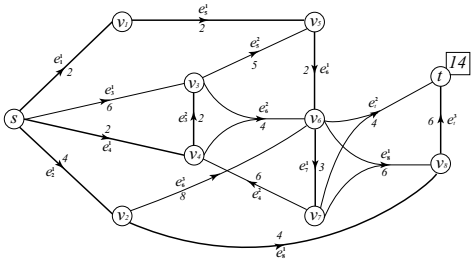
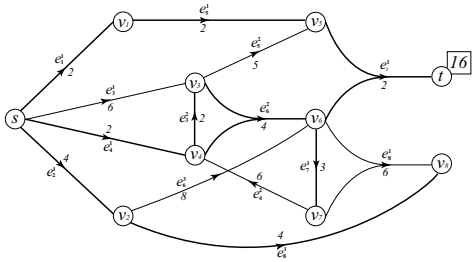


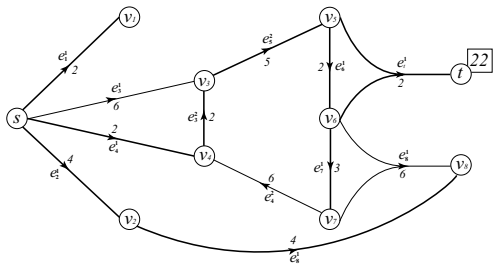

Figure 10: The branching tree $\tilde{T}$ of $\mathcal{H}$ and its corresponding subhypergraphs (subhypergraph $\mathcal{H}_4$ is not shown). The hyperarcs defining the minimal hyperpaths have been emphasized.




(a) Subhypergraph $\mathcal{H}_1$.

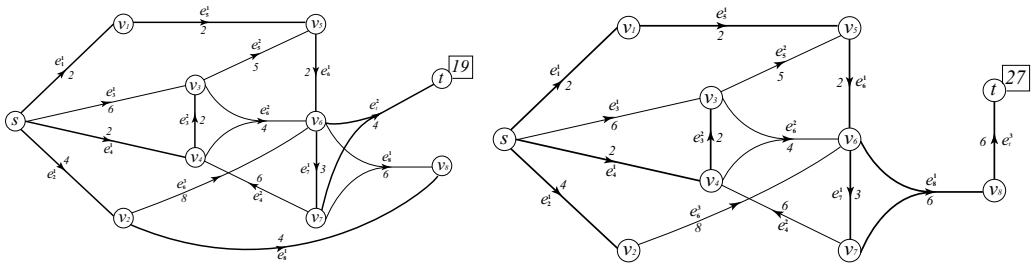



(b) Subhypergraph $\mathcal{H}_2$.




(c) Subhypergraph $\mathcal{H}_3$.

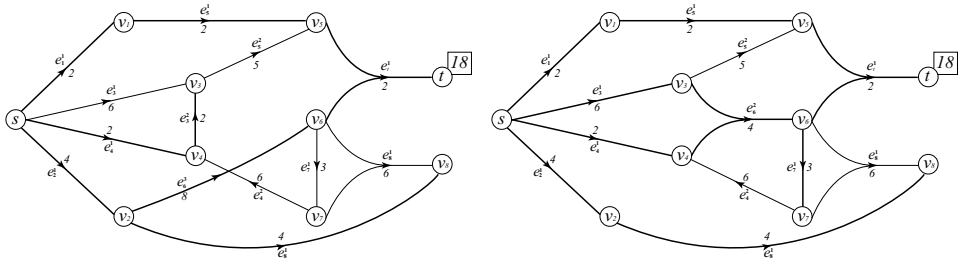
Figure 11: The subhypergraphs $\mathcal{H}_1, \mathcal{H}_2$ and $\mathcal{H}_3$ of $\mathcal{H}$ with their corresponding minimal hypertrees emphasized.




(a) Subhypergraph $\mathcal{H}_{11}$.

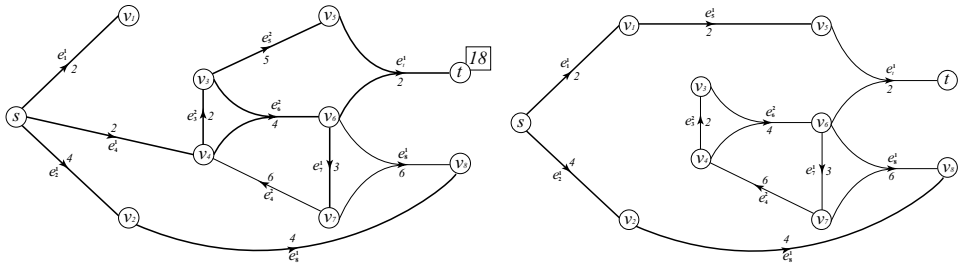
(b) Subhypergraph $\mathcal{H}_{12}$.

Figure 12: The subhypergraphs $\mathcal{H}_{11}$ and $\mathcal{H}_{12}$ of $\mathcal{H}_1$ with their corresponding minimal hypertrees emphasized.




(a) Subhypergraph $\mathcal{H}_{21}$.

(b) Subhypergraph $\mathcal{H}_{22}$.




(c) Subhypergraph $\mathcal{H}_{23}$.

(d) Subhypergraph $\mathcal{H}_{24}$.

Figure 13: The subhypergraphs $\mathcal{H}_{21}$,...,$\mathcal{H}_{24}$ of $\mathcal{H}_2$ with their corresponding minimal hypertrees emphasized.

their corresponding minimal hypertree are shown in Figure 12 and if we add these solutions to *List* we have

$$List = \{(\mathcal{H}_{11}, \pi_{11}), (\mathcal{H}_{12}, \pi_{12}), (\mathcal{H}_2, \pi_2), (\mathcal{H}_3, \pi_3)\}$$

Again, pick one with lowest weight, i.e. $(\mathcal{H}_2, \pi_2)$ with $W(t) = 16$. The ordered branching set becomes

$$V = (t, 6, 3, 5, 4, 1)$$

which corresponds to 6 subhypergraphs. Three of them have no hyperpath from $s$ to $t$. By removing $p(t)$, for instance, we do not have arcs in the backward star of $t$ and therefore no

hyperpath $\pi_{st}$. Four of the subhypergraphs of $\mathcal{H}_2$ are shown in Figure 13 with their minimal hypertree. Note that for $\mathcal{H}_{24}$ we do not have a hyperpath $\pi_{st}$, so we do not add it to *List* which becomes:

$$List = \{(\mathcal{H}_{21}, \pi_{21}), (\mathcal{H}_{22}, \pi_{22}), (\mathcal{H}_{23}, \pi_{23}), (\mathcal{H}_{11}, \pi_{11}), (\mathcal{H}_{12}, \pi_{12}), (\mathcal{H}_3, \pi_3)\}$$

Now, we just have to pick the one with lowest weight and we have found the $4'th$ shortest hyperpath of $\mathcal{H}$. Hypergraph $\mathcal{H}$ actually has 24 different hyperpaths. ■

Example 3 leads to the following simple observation.

**Observation 1** If the forward star of a node $v \neq t$ is empty, this node will never be a node in a hyperpath $\pi_{st}$. Therefore all nodes $v$, where $|FS(v)| = 0$, can be removed from every subhypergraph $\tilde{\mathcal{H}}$ without removing any of the hyperpaths $\pi_{st}$ contained in $\tilde{\mathcal{H}}$. This can be done recursively, meaning that if removal of a node makes the forward star of another node empty, this node can be removed too. The same rule is valid for the backward star of all nodes $v \neq s$.

Consider for instance node $v_8$ in Figure 11 which can be removed from subhypergraph $\mathcal{H}_2$ and $\mathcal{H}_3$, since the forward star is empty.

## 3.1 Better branching rules

In the above we used an ordered branching set $V$ of the nodes in the hyperpath to specify the ordering of the levels in the branching tree $\tilde{T}$. This ordering can be chosen randomly but why not choose an ordering where we can use information already calculated.

### 3.1.1 Backward branching

When using a shortest hyperpath procedure, we find a minimal hypertree $\mathcal{T}_s$ defined by predecessor function $p$ where the minimal hyperpath $\pi$ is a subhypergraph of $\mathcal{T}_s$. Therefore, when the procedure ends, each node $v$ contains a label of the minimal weight to node $v$ and another label containing the predecessor hyperarc. We here can assume that $FS(t) = \emptyset$ because a hyperarc $e \in FS(t)$ will never be used in a hyperpath $\pi_{st}$. It is therefore possible to find a valid ordering of $\mathcal{T}_s$ :

$$V_{\mathcal{T}_s} = (v_1 = s, v_2, ..., v_{n-1}, v_n = t)$$

with $t$ as the last node in the ordering. Note that the valid ordering can be found as the order in which we pick the nodes in a labelsetting version of procedure SBT, provided that Dijkstra's principle and assumption (2.3) is fulfilled. From Theorem 3 we now have that if subhypergraph $\tilde{\mathcal{H}}$ is obtained from $\mathcal{H}$ by removing some hyperarcs from $BS(v_i), ..., BS(v_n)$ we still have the same minimal hypertree to the nodes $v_1, ..., v_{i-1}$ for $\tilde{\mathcal{H}}$. Now, if we order the levels of our branching tree $\tilde{T}$ in the opposite of $V_{\mathcal{T}_s}$, then for each subhypergraph $\tilde{\mathcal{H}}_i$ we only have to find a new minimal hyperpath to the node $u_i$ and the nodes after it in $V_{\mathcal{T}_s}$ (see Figure 14). That is we can use the labels in node $v_1, ..., v_{l-1}$ calculated before. Another good property of using the opposite valid ordering of the minimal hypertree is, that we do

$$V_{\mathcal{T}_s} = (\underbrace{v_1, v_2, ..., v_{l-1}}_{same\ hp}, \overset{u_i}{\underbrace{\overset{\downarrow}{v_l}\ , ..., v_{n-1}, v_n}_{recalculate}})$$

Figure 14: Use of information in $\mathcal{T}_s$.

not have to branch on nodes $v$ in the hyperpath where $|BS(v)| = 1$. This is due to the fact that, if the opposite valid ordering is used, we force the tail nodes of each hyperarc we fix to be nodes in the minimal hyperpath of the subhypergraph. Therefore next removing the hyperarc from a tail node $v$ with $|BS(v)| = 1$, leaves the backward star empty and therefore no hyperpath $\pi_{st}$ can exist. The above leads to the following branching rule.

**Branching Rule 2 (Backward branching)** Let predecessor function $p$ define the minimal hypertree $\mathcal{T}_s$ of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ which contains the minimal hyperpath $\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$. Moreover, let $V_{\mathcal{T}_s}$ denote a valid ordering of $\mathcal{T}_s$.

$$V_{\mathcal{T}_s} = (v_1 = s, v_2, ..., v_{n-1}, v_n = t)$$

Now, scan $V_{\mathcal{T}_s}$ backwards from node $v_n$ to node $v_1$ and add node $v_i$ to the ordered branching set $\tilde{V}$ if

$$v_i \in \mathcal{V}_\pi \text{ and } |BS(v_i)| > 1$$

When $V_{\mathcal{T}_s}$ has been scanned, we now have an ordered branching set

$$V = \left(v_{l_1}, ..., v_{l_p}\right)$$

where the last node added from $V_{\mathcal{T}_s}$ is last in $V$. We now create subhypergraph $\mathcal{H}_i$, $1 \leq i \leq p$ by branching on node $v_{l_i}$.

**Example 3** (continued) The ordering we used for the branching tree $\tilde{T}$ was exactly the opposite of a valid ordering of the minimal hypertree. A valid ordering of the minimal hypertree of $\mathcal{H}$ in Figure 9 could be

$$V_{\mathcal{T}_s} = (s, 2, 8, 4, 3, 1, 5, 6, 7, t)$$

and the ordering of the branching tree $\tilde{T}$ was

$$V = (t, 6, 5, 1)$$

Here $|BS(1)| = 1$, so it is not necessary to branch on node 1, since fixing the backward star of node $t, 6$ and 5 forces node $v_1$ to be a node in the hyperpath and therefore removing $e_1^1$ will disconnect $s$ from $t$ (see Figure 11). Observe that this is a consequence of the branching rule used. If we choose an ordering

$$V = (t, 1, 6, 5)$$

we have to branch on node 1. The same is the case when we have to find subhypergraphs of $\mathcal{H}_2$ in Figure 11. Here the ordering was

$$V = (t, 6, 3, 5, 4, 1)$$

and if we remove the nodes $v$ with $|BS(v)| = 1$ the ordering becomes

$$V = (6, 3, 5, 4)$$

These subhypergraphs are shown in Figure 13. Note that it is still possible to have subhypergraphs where $t$ is not hyperconnected to $s$.
Subhypergraph $\mathcal{H}_3$ was obtained from $\mathcal{H}$ by fixing $BS(t)$ and $BS(6)$ and deleting a hyperarc from $BS(5)$. Therefore all nodes before node 5 in the valid ordering $V_{\mathcal{T}_s}$ will have the same minimal hyperpath in $\mathcal{H}_3$ as in $\mathcal{H}$ (see Figure 11).
It is of course best to find a valid ordering of the minimal hypertree where the nodes in $\pi_{st}$ is as far back in the ordering as possible. Another valid ordering of the minimal hypertree of $\mathcal{H}$ could be

$$V_{\mathcal{T}_s} = (s, 1, 5, 6, t, 7, 2, 8, 4, 3)$$

But with this ordering we cannot use much information. ■

Given Branching Rule 2, we now have a faster $k'th$ procedure shown in Procedure 7.

**Precondition:** Same as in Procedure 6 on page 14.

**Postcondition:** Same as in Procedure 6 on page 14.

**Initialization:** Use an SBT procedure to calculate a minimal hyperpath $\pi \in \Pi$ and set $List = \{(\mathcal{H}, \pi)\}$

1 **for** ($i = 1$ to $k$) **do** pick $\left(\pi^i, \mathcal{H}^i\right) = \arg\min_{(\tilde{\pi}, \tilde{\mathcal{H}}) \in List} \{W(\tilde{\pi})\}$ and set $Out := Out \cup \left\{\pi^i\right\}$
2 **for** ($j = 1$ to $q$) **do** create subhypergraph $\mathcal{H}_j$ using Branching Rule 2.
3 calculate $\pi_j = \min_{\pi \in \Pi_{\mathcal{H}_j}} \{W(\pi)\}$ using an modified SBT procedure which use information
4 set $List := List \cup \{(\pi_j, \mathcal{H}_j)\}$
5 **end for**
6 **end for**

**Procedure 7:** Finding the $k$ shortest hyperpaths using backward branching.

## 3.2 Using lower bounds to reduce computation time

In the above we assumed that $k$ was known and each time we picked a hypergraph from the candidate set, we calculated a new solution for each of its subhypergraphs. But when we search for nondominated solutions in the bicriterion procedures in Section 4, $k$ is not known. We search until we reach an upper bound, which may be lowered during the procedure, and then stop. So, if possible we want to make computations as late as possible, i.e. when we pick a hypergraph from the candidate set, we do not want to calculate a minimal hyperpath for each of its subhypergraphs. Instead, we calculate a lower bound $\underline{W}$ of the shortest hyperpath in each subhypergraph and add this solution to the candidate set. It is obviously crucial that this lower bound is as close as possible to the true value and that the computation of lower bounds is faster than calculating the weight of the shortest hyperpath. We now pick the hypergraph $\tilde{\mathcal{H}}$ from the candidate set with minimal lower bound and have two cases:

1. If the lower bound is over the upper search bound we stop because all solutions needed have been found.
2. Otherwise we calculate the right minimal weight hyperpath $\pi$ of $\tilde{\mathcal{H}}$ and create subhypergraphs $\mathcal{H}_1, ..., \mathcal{H}_q$ by using Branching Rule 2. Furthermore, if the lower bound for subhypergraph $\tilde{\mathcal{H}}_i$ is under the upper bound we add $\tilde{\mathcal{H}}_i$ to the candidate set, otherwise not.

Note that we branch on the minimal hyperpath $\pi$ which has the minimal lower bound. Therefore we do not find the $k$ shortest hyperpaths in the right order, i.e. we could find the $1'st$, $2'nd$, $4'th$, $3'rd$, $6'th$, .... but this does not matter, if you just want to find all solutions up to a certain upper bound. We could evidently be compelled to calculate the minimal weight hyperpath $\pi$ of a subhypergraph where $W(\pi)$ is over the upper bound, if the lower bound is weak. But if the lower bound is close to the right value, this would be rare. A lower bound can be found using the following theorem

**Theorem 5** Let predecessor function $p$ define the minimal hypertree $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_{\mathcal{T}_s})$ of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and let $W(u)$ denote the optimal weight in each node $u \in \mathcal{N}$. Moreover, let $V = (u_1, ..., u_p)$ denote the ordered branching set found by using Branching Rule 2 and let $\mathcal{H}_i$ denote the subhypergraph of $\mathcal{H}$ found by branching on node $u_i$. Then

$$\underline{W}(u_i) = \min_{e \in BS(u_i) \setminus p(u_i)} F(e) + w(e) \tag{3.2}$$

is a lower bound on the weight of the minimal hyperpath from $s$ to $u_i$ in $\mathcal{H}_i$ where $F(e)$ denotes the chosen weighting function.

**Proof** Let $\underline{e}$ denote the optimal hyperarc of equation (3.2). Since the optimal weights of node $u \in T(\underline{e})$ in $\mathcal{H}_i$ will never be less than the optimal weights of node $u \in T(\underline{e})$ in $\mathcal{H}$,

**Precondition:** Let $List$ be the candidate set, $ub$ the upper bound, $Out$ the outputlist and let $lb$ denote the lower bound for subhypergraph $\tilde{\mathcal{H}}$. Moreover, let $q$ denote the number of elements in the ordered branching set of $\bar{\mathcal{H}}$.

**Postcondition:** The outputlist $Out$ has been filled with the hyperpaths which have weight under upper bound $ub$.

**Initialization:** Set $lb = 0$, $List = \{(\mathcal{H}, 0)\}$ and $\bar{\mathcal{H}} = \mathcal{H}$

1 **while** $(lb \leq ub)$ **do** use an SBT procedure to calculate the minimal hyperpath $\pi$ of $\bar{\mathcal{H}}$
2  **if** $(W(\pi) \leq ub)$ **then** $Out := Out \cup \{\pi\}$
3  **for** $(i = 1$ to $q)$ **do** create subhypergraph $\mathcal{H}_i$ using Branching Rule 2.
4  calculate $lb_i$ using Theorem 5 on the preceding page
5  **if** $(lb_i \leq ub)$ **then** set $List := List \cup \{(\mathcal{H}_i, lb_i)\}$
6  **end for**
7  pick $(\bar{\mathcal{H}}, lb) = \arg\min_{(\tilde{\mathcal{H}}, \tilde{lb}) \in List} \left\{\tilde{lb}\right\}$
8 **end while**

**Procedure 8:** Finding all shortest hyperpaths under an upper bound

we have that $\underline{W}(u_i)$ using the optimal weights of $\mathcal{H}$ is a lower bound on the actual minimal weight. ■

We can now calculate the lower bound weight of node $t$ by changing $W(u_i)$ to $\underline{W}(u_i)$ and calculate new weights for the nodes succeeding $u_i$ in the valid ordering $V_{\mathcal{T}_s}$. Procedure 8 finds all hyperpaths with a weight under an upper bound $ub$.
In Procedure 8 we do not calculate the $k'th$ hyperpaths in the right order, but if this is necessary we can do the following. Let $W(\pi)$ be the weight of the minimal hyperpath of $\bar{\mathcal{H}}$ on line 1 in Procedure 8 and let $(\underline{\mathcal{H}}, \underline{lb})$ be the element with minimal lower bound currently in $List$. We now have two cases

1. $W(\pi) \leq \underline{lb}$ then $\pi$ is minimal and we add it to $Out$.
2. $W(\pi) > \underline{lb}$ then just reinsert $\left(\bar{\mathcal{H}}, W(\pi)\right)$ in $List$ and go directly to line 7.

By doing the above, we only add a hyperpath to $Out$ when it is minimal for all the subhypergraphs currently in $List$ and otherwise we do not branch on it, but pick the subhypergraph with a better lower bound. So we pick the hyperpaths in the right order.

## 3.3 The acyclic case

Assume that $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is acyclic. Consequently, according to Theorem 1, a valid ordering of the nodes of $\mathcal{H}$ exists.

$$V_{\mathcal{H}} = (v_0 = s, v_1, ..., v_q, ..., v_n = t)$$

This valid ordering can now be used in Branching Rule 2 and since $\mathcal{H}$ is acyclic, we can find the minimal hyperpath of $\mathcal{H}$ using an acyclic SBT procedure which use a backward star representation, i.e. no forward representation is needed in the acyclic case. Moreover, in the acyclic case the lower bounds calculated in Section 3.2 are actually not lower bounds but the true weight of a minimal hyperpath.

**Theorem 6** Let predecessor function $p$ define a minimal hypertree $\mathcal{T}_s$ of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and let $W(v)$ denote the optimal weight in each node $v \in \mathcal{V}$. Moreover, let $V = (u_1, ..., u_p)$ denote the ordered branching set found by using Branching Rule 2 and let $\mathcal{H}_i$ denote the subhypergraph of $\mathcal{H}$ found by branching on node $u_i$. Then

$$\tilde{W}(u_i) = \min_{e \in BS(u_i) \setminus p(u_i)} F(e) + w(e) \tag{3.3}$$

**Precondition:** Same as in Procedure 6 on page 14.

**Postcondition:** Same as in Procedure 6 on page 14.

**Initialization:** Use an acyclic SBT procedure to calculate a minimal hyperpath $\pi \in \Pi$ of $\mathcal{H}$ and set $List = \{(\mathcal{H}, \pi)\}$.

1 **for** ($i = 1$ to $k$) **do** pick $\left(\pi^i, \mathcal{H}^i\right) = \arg\min_{(\tilde{\pi},\tilde{\mathcal{H}})\in List} \{W(\tilde{\pi})\}$ and set $Out := Out \cup \left\{\pi^i\right\}$
2 **for** ($j = 1$ to $q$) **do** create subhypergraph $\mathcal{H}_i$ using Branching Rule 2 with valid ordering $V_{\mathcal{H}}$
3 calculate $\pi_i = \min_{\pi\in\Pi_{\mathcal{H}_i}} \{W(\pi)\}$ using Theorem 6 on the preceding page.
4 set $List := List \cup \{(\pi_i, \mathcal{H}_i)\}$
5 **end for**
6 **end for**

**Procedure 9:** Finding the $k$ shortest hyperpaths (acyclic case).

is the weight of a minimal hyperpath $\pi_{su_i}$ from $s$ to $u_i$ of $\mathcal{H}_i$ where $F(e)$ denotes the chosen weighting function using $W(v)$ for $v \in T(e)$. Moreover, predecessor function $\tilde{p}$

$$\tilde{p}(v_j) = \begin{cases} e & v_j = u_i \\ p(v_j) & \text{otherwise} \end{cases}$$

where $e$ is the predecessor hyperarc chosen in equation (3.3), defines a hypertree of $\mathcal{H}_i$ which contains the minimal hyperpath $\pi_{su_i}$.

**Proof** Given $\mathcal{H}$ acyclic and $V_{\mathcal{T}_s} = (v_0, ..., v_p = u_i, ..., v_n)$, we have that $T(e) \in \{v_1, ..., v_{p-1}\}$ and therefore the minimal weights in $T(e)$ for $\mathcal{H}_i$ is the same as in $\mathcal{H}$ according to Theorem 3. So $\tilde{W}(u_i)$ is the minimal weight of the hyperpath $\pi_{su_i}$ contained in the hypertree, defined by the predecessor function $\tilde{p}$ where

$$\tilde{p}(v_j) = \begin{cases} e & v_j = u_i \\ p(v_j) & \text{otherwise} \end{cases}$$

That is $\tilde{W}(u_i)$ is not a lower bound but the actual minimal weight. ■

Using Theorem 6, the minimal weight of subhypergraph $\mathcal{H}_i$ can be found by calculating the weight $W(t)$ in the hypertree defined by $\tilde{p}$. So, to calculate the $k'th$ shortest hyperpath, we only have to calculate the shortest hyperpath of $\mathcal{H}$ and afterwards make trivial computations using Theorem 6 and Branching Rule 2. The procedure is stated in Procedure 9.

# 4 Bicriterion shortest hyperpaths

Let $\mathcal{H}$ be a hypergraph where each hyperarc $e$ is assigned two real weights

$$w_i(e) \qquad i = 1, 2$$

and let $W_i(\pi)$ denote the corresponding additive weighting function of hyperpath $\pi$ using weights $w_i(e)$. Moreover, if the value function is considered, we define nonnegative multipliers for each hyperarc $e$ and node $v \in T(e)$

$$a_i^e(v) \qquad i = 1, 2,\ v \in T(e)$$

In this paper we assume that $a_1^e(v) = a_2^e(v)$, i.e. the multipliers are the same for both weighting functions. Let $\Pi$ denote the set of hyperpaths $\pi_{st}$ from $s$ to $t$, i.e.

$$\Pi = \{\pi \subseteq \mathcal{H} \,|\pi \text{ has root } s \text{ and destination } t\}$$

We now wish to solve *the bicriterion shortest hyperpath problem* (bi-SBT)

$$\min_{\pi\in\Pi} W(\pi) = (W_1(\pi), W_2(\pi)) \tag{4.1}$$

That is to find hyperpaths from a given root $s$ to a given node $t$, where the two weights are minimal in the sense that we cannot improve one weight without worsening the other. We consider the following cases of weighting functions: value/value and distance/distance. Let us follow the terminology in Skriver [25]. Given a hyperpath $\pi$ we say

**Definition 4** A hyperpath $\pi \in \Pi$ is *efficient* if and only if

$$\nexists path\ \widetilde{\pi} \in \Pi : W_1(\tilde{\pi}) \leq W_1(\pi) \text{ and } W_2(\tilde{\pi}) \leq W_2(\pi) \tag{4.2}$$

with at least one strict inequality. Otherwise $\pi$ is *inefficient.*

Efficient hyperpaths are defined in decision space and their counterpart is points in criterion space

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

**Definition 5** A point $W(\pi) \in \mathcal{W}$ is a *nondominated* criterion point if and only if $\pi$ is an efficient hyperpath. Otherwise $W(\pi)$ is a *dominated* criterion point.

The criterion points can be partitioned into two kinds, namely supported and unsupported. The supported ones can be further subdivided into supported extreme and supported nonextreme. Let us define

$$\begin{aligned} \Pi_{Eff} &= \{\pi \in \Pi |\ \pi \text{ is efficient}\} \\ \mathcal{W}_{Eff} &= \left\{ W(\pi) \in \mathbb{R}^2 \mid \pi \in \Pi_{Eff} \right\} \\ \mathcal{W}^{\geq} &= conv\left( \mathcal{W}_{Eff} \oplus \left\{ \mathbf{w} \in \mathbf{R}^2 \mid \mathbf{w} \geq 0 \right\} \right) \end{aligned}$$

**Definition 6** $W(\pi) \in \mathcal{W}_{Eff}$ is a *supported* nondominated criterion point if $W(\pi)$ is on the boundary of $\mathcal{W}^{\geq}$ denoted $\mathcal{W}^{=}$. Otherwise $W(\pi)$ is an *unsupported* nondominated criterion point.

Notice that unsupported nondominated criterion vectors are dominated by a convex combination of other nondominated criterion vectors [27].

**Definition 7** A supported point $W(\pi)$ is a *supported extreme* nondominated criterion point if $W(\pi)$ is an extreme point of $\mathcal{W}^{\geq}$. Otherwise $W(\pi)$ is an *supported nonextreme* nondominated criterion point.

It is well-known that a set of nondominated points $\Phi = \left\{ W^1, W^2, ..., W^l \right\}$ can be ordered in the following way:

$$W_1^1 < W_1^2 < ... < W_1^l \qquad W_2^1 > W_2^2 > ... > W_2^l$$

We call $\Phi$ an *ordered nondominated set.* It is easy to check if a new point is dominated in $\Phi$. Procedure 10 checks and adds $W$ to $\Phi$ if $W$ is dominated and deletes new dominated points from $\Phi$. Line 3 searches the set until $W_1^i > W_1$ and line 4 checks if $W$ is dominated. If not, we have 2 options: If $W_1^{i-1} = W_1$, we have that $W^{i-1}$ is dominated and we insert $W$ instead on line 5. Otherwise we insert the point between $W^{i-1}$ and $W^i$. Next on line 7 we remove all points dominated by $W$.

**Example 4** The criterion space is illustrated in Figure 15. Here $\mathcal{W}^{=}$ is the border drawn with the hard lines and $\mathcal{W}^{\geq}$ the union of $\mathcal{W}^{=}$ and the area above. $W^1$ is the point which has minimal weight w.r.t. weight two when weight one is fixed to its minimal weight. We call $W^1$ the *upper/left point* and likewise $W^8$ the *lower/right point* of the criterion space. All nondominated points will be inside the square defined by $W^1$ and $W^8$, therefore $W^9$ is dominated. The points $W^2, W^3$ and $W^6$ are all supported nondominated points; furthermore,

**Precondition:** Let $\Phi$ denote an ordered nondominated set and let $W = (W_1, W_2)$ be a point.

**Postcondition:** $W$ has been added to $\Phi$ if it is nondominated and dominated points in $\Phi$ have been removed.

```
1 procedure insert(W, Φ)
2     set i = 1
3     while (W_1^i ≤ W_1 or i = l) do i := i + 1
4     if (W_2^{i-1} ≤ W_2) then stop (W is dominated)
5     if (W_1^{i-1} = W_1) then W^{i-1} := W
6     else insert W between W^{i-1} and W^i
7     while (W_2^i ≥ W_2) do remove W^i from Φ and set i := i + 1
8 end procedure
```

**Procedure 10:** Inserting a point $W$ in an ordered nondominated set $\Phi$.

$W^2$ and $W^6$ are supported extreme nondominated points. All supported extreme points define the *triangles* also called gaps (dashed lines) in which it is possible to find nondominated points. Therefore all points, e.g. $W^7$, outside the triangles will be dominated. If we look at the triangle defined by $W^2$ and $W^6$ we have that $W^4$ is a nondominated point and $W^5$ is dominated. ■

## 4.1 Finding efficient hyperpaths using a simple $k'th$ shortest hyperpath method.

In this section we present a simple $k'th$ best procedure. For digraphs a similar procedure is presented in Climaco and Martins [5]. The idea is simple, first find the lower/right point $\check{W} = \left(\check{W}_1, \check{W}_2\right)$. This gives an upper bound on weight one. We then use a $k'th$ best procedure on weight one to find all nondominated points below the upper bound $\check{W}_1$. Note that the lower/right point can normally not always be found by finding the shortest hyperpath w.r.t.[5] the second criteria because there can be more than one hyperpath which

[5]With respect to.

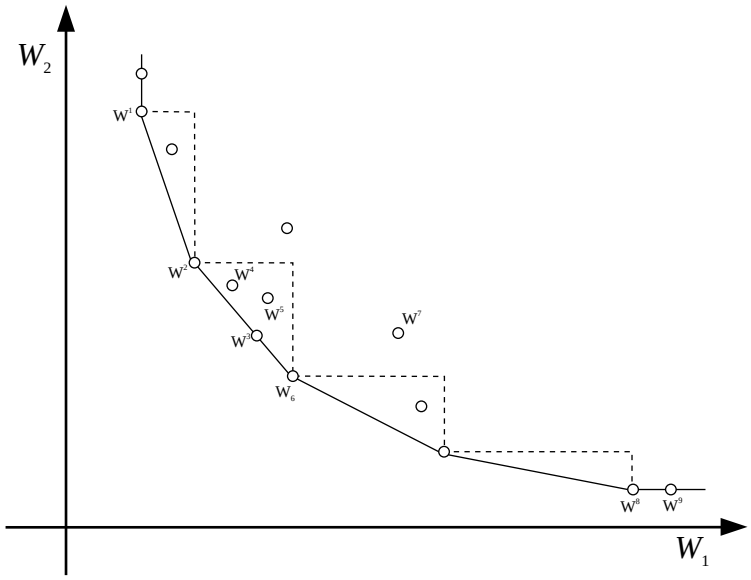



Figure 15: Criterion space for different hyperpaths.

**Precondition:** Let $\Phi$ denote an ordered nondominated set and let $W^k = \left(W_1^k, W_2^k\right)$ denote the $k'th$ solution of a shortest hyperpath procedure calculating the $k'th$ shortest hyperpath with respect to the first criteria.

**Postcondition:** $\Phi$ contains all nondominated points.

**Initialization:** Use an SBT procedure to find the shortest hyperpath $\check{\pi}$ w.r.t. the second criteria and set $w_2^{\min} = W_2(\check{\pi})$. Compute $W^1$ and set $W^{Cand} = W^1$, $lb = W_1^1$, $\Phi = \emptyset$.

1 **while** $(W_2^k \neq W_2^{\min})$ **do** set $k := k + 1$ and compute $W^k$.
2 **if** $(W_1^k = lb$ and $W_2^k < W_2^{Cand})$ **then** set $W^{Cand} = W^k$
3 **if** $(W_1^k > lb$ and $W_2^k < W_2^{Cand})$ **then** set $\Phi = \Phi \cup \left\{W^{Cand}\right\}$, $lb = W_1^k$, $W^{Cand} = W^k$
4 **end while**
5 $\Phi = \Phi \cup \left\{W^{Cand}\right\}$ (add the lower/right point)

**Procedure 11:** A simple $k'th$ best search procedure.

has minimal weight w.r.t. the second criteria. But it is actually not necessary to find the right upper bound because if we store the minimal weight $\check{W}_2$ instead, we know that the first point $W^i = \left(W_1^i, W_2^i\right)$ found with $W_2^i = \check{W}_2$ is the lower/right point and stop. The procedure is stated in Procedure 11. The candidate point $W^{Cand}$ is always the lowest currently nondominated point found w.r.t. the second weight (line 2) when the $W_1$ is fixed. If a new candidate is found which have a higher first weight and a lower second weight than the current candidate, the current candidate must be a nondominated point and we add it to the nondominated set $\Phi$. The new point now becomes the new candidate (line 3).
For digraphs, Procedure 11 seems to be slow, since there are to many paths to search [19]. This is also the case for hypergraphs, as we will see in Section 6.

## 4.2 Finding efficient hyperpaths using a two-phases approach

In this section we consider a two-phases approach where the search for nondominated points are split into two-phases. Phase one finds supported extreme nondominated points defining the triangles in which nonsupported nondominated points may be found. Phase two then searches the triangles using a $k'th$ best procedure.

### 4.2.1 Phase 1: Finding supported nondominated solutions

We present a NISE like algorithm [6] for finding supported extreme nondominated points. These can be found by parametrizing the criteria vector. Let $f : (\Pi, \mathbb{R}_+) \longrightarrow \mathbb{R}_+$ denote the function

$$f(\pi, \lambda) = W_1(\pi)\lambda + W_2(\pi)$$

Since the number of hyperpaths in $\Pi$ is finite, we have that $f(\pi, \lambda)$ for fixed $\pi$ defines a line with slope $W_1(\pi)$, intersection $W_2(\pi)$, and the number of lines is finite. Using the parametric function $f(\pi, \lambda)$ we wish to solve

$$f(\lambda) = \min_{\pi \in \Pi} f(\pi, \lambda) \tag{4.3}$$

i.e. we want to find a minimum weight hyperpath $\pi_\lambda = \arg\min_{\pi \in \Pi} f(\pi, \lambda)$ given a fixed $\lambda$. We illustrate the approach with the following example

**Example 5** The criterion space and its corresponding parametric space is shown in Figure 16. Here each point $W(\pi)$ corresponds to a line with slope $W_1(\pi)$ and intersection $W_2(\pi)$. For each fixed $\lambda$ we have a minimal line and the lower envelope of the lines defines $f(\lambda)$ which is a nondecreasing piecewise linear function with breakpoints $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. Each piece of $f(\lambda)$ corresponds to a supported extreme nondominated point and each breakpoint $\lambda_i$ corresponds to a value where the two adjacent supported extreme nondominated points

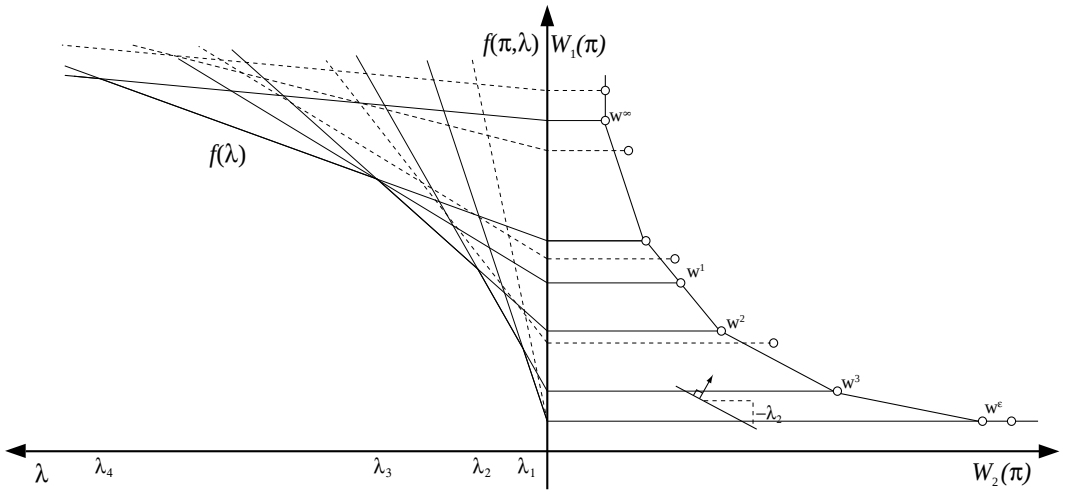



Figure 16: The criterion space and its corresponding parametric space.

have the same minimal parametric weight. If e.g. $\lambda = \lambda_2$ then $W^2$ and $W^3$ have same minimal parametric weight, i.e. fixing $\lambda$ to $\lambda_2$ corresponds to searching for the first point in the direction of the normal of the line shown in Figure 16. Furthermore, we have that $W^1$, which is a supported nonextreme nondominated point, touches $f(\lambda)$ in the breakpoint of its adjacent supported extreme points. Note that the whole set of nondominated points is normally nonconvex, i.e. when using the parametric method, we only find points on the frontier $\mathcal{W}^{=}$ of the criterion space. This can be seen in Figure 16 where all nondominated points inside the triangles corresponds to dashed lines which all lie above $f(\lambda)$. The upper/left and lower/right point can be found by solving problem (4.3) with $\lambda$ very high and $\lambda$ close to zero respectively. Therefore the upper/left and lower/right point are often referred to as $W^{\infty}$ and $W^{\varepsilon}$. ■

Solving problem (4.3) for digraphs, corresponds to solving a shortest path problem with weight $w_1(e)\,\lambda + w_2(e)$ on each arc. Therefore a NISE like procedure for digraphs first finds the upper/left and lower/right point. Then calculate the slope of the line between the two points and search in that direction. If a new supported extreme nondominated point is found, we get two new triangles which are searched like before; otherwise we stop. Notice that it is possible to find a supported nonextreme nondominated point $\tilde{W}$ if the search direction is equal to the normal of the line of two adjacent supported extreme nondominated points $\tilde{W}^1$ and $\tilde{W}^2$. Hence $\tilde{W}^1, \tilde{W}^2$ and $\tilde{W}$ all have same minimal parametric weight, and the point found depends on the SBT procedure.

Let *the parametric weighting function* for a hyperpath $\pi$ denote the weighting function

$$W_\lambda(v) = \begin{cases} w_\lambda(e_\pi(v)) + F(e_\pi(v)) & v \in \mathcal{V} \setminus \{s\} \\ 0 & v = s \end{cases} \tag{4.4}$$

where $e_\pi(v)$ denotes the predecessor of node $v$ and $w_\lambda(e) = w_1(e)\,\lambda + w_2(e)$. Using the parametric weighting function does not always give the right solution to (4.3) when we consider hypergraphs. We look at the following cases: Two value weighting functions and two distance weighting functions.

**Precondition:** Let $\left(W^{+}, W^{-}\right)$ define a search direction, and let $\Phi$ be an ordered nondominated set. Given $W \in \Phi$ let $W^{next}$ denote the point following $W$ in $\Phi$, if $W$ is the last element in $\Phi$ then $W^{next} := null$.

**Initialization:** Use an SBT procedure to find the upper/right point $W^{\infty}$ and the lower/left point $W^{\varepsilon}$.

1 **if** $(W^{\infty} = W^{\varepsilon})$ **then** stop (there is only one nondominated solution)
2 **else** set $\Phi = \{W^{\varepsilon}, W^{\infty}\}$, $W^{+} = W^{\infty}$, $W^{-} = W^{\varepsilon}$
3 **while** $(W^{+} \neq W^{\varepsilon})$ **do** set $\lambda = \left|(W_2^{-} - W_2^{+})/(W_1^{-} - W_1^{+})\right|$, $W_{\lambda}^{+} = W_1^{+}\lambda + W_2^{+}$
4 **for** $(e \in \mathcal{E})$ **do** set $w_{\lambda}(e) = w_1(e)\,\lambda + w_2(e)$
5 find the minimal hyperpath $\pi$ w.r.t. the parametric weighting function
6 **if** $(f(\pi, \lambda) < W_{\lambda}^{+})$ **then call** procedure insert$(W(\pi), \Phi)$
7 $W^{-} = W(\pi)$
8 **end if**
9 **else** set $W^{+} = W^{-}$, $W^{-} = W^{+next}$
10 **end while**

**Procedure 12:** Finding all supported extreme nondominated points (value/value).

**Two value weighting functions** We consider two value weighting functions, i.e. given a hyperpath $\pi = (\mathcal{V}_{\pi}, \mathcal{E}_{\pi})$ from $s$ to $t$ the weighting function $W_i$ of $\pi$ is

$$W_i(v) = \begin{cases} w_i(e_{\pi}(v)) + \sum\limits_{u \in T(e_{\pi}(v))} a^{e_{\pi}(v)}(u)\, W_i(u) & v \in \mathcal{V}_{\pi} \setminus \{s\} \\ 0 & v = s \end{cases}$$

for $i = 1, 2$. If we want to solve SBT using the parametric weighting function, we have to solve the following recursive equations

$$W_{\lambda}(v) = \begin{cases} \min\limits_{e \in BS(v)} \left\{ w_{\lambda}(e) + \sum\limits_{u \in T(e)} a^{e}(u)\, W_{\lambda}(u) \right\} & v \in \mathcal{V} \setminus \{s\} \\ 0 & v = s \end{cases} \tag{4.5}$$

Since finding the shortest hyperpath w.r.t. the value function $W_i$ can be formulated as an LP problem, we have that solving (4.5) corresponds to solving an LP problem with the same constraints, but with an objective function cost for hyperarc $e$ on $w_1(e)\,\lambda + w_2(e)$ instead of $w_i(e)$. Therefore the minimal hyperpath $\pi$ w.r.t. the parametric weighting function satisfies that $W_{\lambda}(\pi) = W_1(\pi)\,\lambda + W_2(\pi)$ i.e. solving (4.5) gives us the solution to (4.3).
A NISE like procedure for two value functions which finds all supported extreme nondominated points, is formulated in Procedure 12. In each step we search in the direction of the normal of the line between $W^{+}$ and $W^{-}$. On line 5 we find a supported nondominated point $W(\pi)$. If $f(\pi, \lambda) < W_{\lambda}^{+}$ then $W(\pi)$ must be a supported extreme nondominated point and we add it to $\Phi$ on line 6. Note that we can insert $W(\pi)$ directly after $W^{+}$ in procedure insert on line 6, since $W(\pi)$ is a new nondominated point.

**Two distance weighting functions** We here consider two distance functions, i.e. given hyperpath $\pi$

$$W_i(v) = \begin{cases} w_i(e_{\pi}(v)) + \max\limits_{u \in T(e_{\pi}(v))} \{W_i(u)\} & v \in \mathcal{V}_{\pi} \setminus \{s\} \\ 0 & v = s \end{cases}$$

for $i = 1, 2$. If we solve SBT using the parametric weighting function we have to solve the following recursive equations

$$W_{\lambda}(v) = \begin{cases} \min\limits_{e \in BS(v)} \left\{ w_{\lambda}(e) + \max\limits_{u \in T(e)} \{W_{\lambda}(u)\} \right\} & v \in \mathcal{V} \setminus \{s\} \\ 0 & v = s \end{cases} \tag{4.6}$$

Because no LP formulation for the distance exists, finding the shortest hyperpath w.r.t. the parametric weighting function does not always give the same solution to (4.3), as can be seen in the following example.

**Example 6** Consider the hypergraph in Figure 17 which contains two hyperpaths and assume that we want to find the shortest hyperpath using the parametric weighting function with $\lambda = 1$. This gives shortest hyperpath $\pi_1$ with predecessor hyperarc $e_4$ in node $v_4$. The weights are $W_\lambda(v_4) = 14 \Rightarrow W_\lambda(t) = 30$. For the same hyperpath, we have that the weights w.r.t. the first and second weight is $W_1(t) = 12$ and $W_2(t) = 20 \Rightarrow f(\pi_1, \lambda) = 32$, hence $W_\lambda(t) \neq W_1(t)\lambda + W_2(t)$.

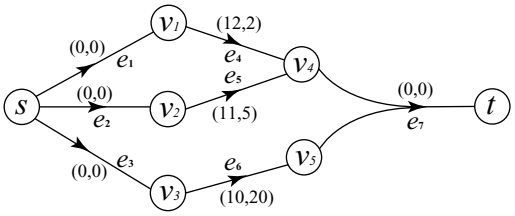



Figure 17: A hypergraph where the minimal hyperpath w.r.t. the parametric weighting function don't give the optimal solution of problem (4.3).

Furthermore, hyperpath $\pi_2$, with $e_5$ as predecessor hyperarc in node $v_4$, gives first and second weight $W_1(t) = 11$ and $W_2(t) = 20 \Rightarrow f(\pi_2, \lambda) = 31$, i.e. $\pi_2$ is the optimal hyperpath to (4.3). ■

Example 6 shows that solving the recursive equations in (4.6) does not always give the right solution to problem (4.3). However, it gives us a lower bound.

**Theorem 7** Let $W_\lambda(\pi)$ be the weight of the shortest hyperpath using the parametric weighting function then $W_\lambda(\pi)$ is a lower bound on the optimal value to (4.3), that is $W_\lambda(\pi) \leq f(\lambda)$.

**Proof** Assume that $\exists \pi \in \Pi : W_\lambda(\pi) \neq f(\lambda)$ and let $\pi_{su}$ denote the subhyperpath of $\pi$ from $s$ to $u$ where $u$ is a node in $\pi$. Given a valid ordering $V = (u, ..., u_p)$ of $\pi$. Let $u_i$ be the first node satisfying $W_\lambda(u_i) \neq f(\pi_{su_i}, \lambda) \Rightarrow W_\lambda(u_j) = f(\pi_{su_j}, \lambda) = W_1(u_j)\lambda + W_2(u_j)$, $\forall j = 1, ..., i-1$. Then

$$\begin{aligned}
W_\lambda(u_i) &= \max_{v \in T(e_\pi(u_i))} \{W_\lambda(v)\} + w_\lambda(e_\pi(u_i)) \\
&= \max_{v \in T(e_\pi(u_i))} \{W_1(v)\lambda + W_2(v)\} + w_1(e_\pi(u_i))\lambda + w_2(e_\pi(u_i)) \\
&\leq \left( \max_{v \in T(e_\pi(u_i))} \{W_1(v)\} + w_1(e_\pi(u_i)) \right) \lambda + \left( \max_{v \in T(e_\pi(u_i))} \{W_2(v)\} + w_2(e_\pi(u_i)) \right) \\
&= W_1(u_i)\lambda + W_2(u_i) = f(\pi_{su_i}, \lambda)
\end{aligned}$$

and hence that $W_\lambda(\pi) \leq f(\pi, \lambda)$. Now let $\hat{\pi}$ denote the shortest hyperpath w.r.t. the parametric weighting function and let $\tilde{\pi}$ be the hyperpath that solves problem (4.3), then

$$W_\lambda(\hat{\pi}) \leq W_\lambda(\tilde{\pi}) \leq f(\tilde{\pi}, \lambda) = f(\lambda)$$

i.e. $W_\lambda(\hat{\pi}) \leq f(\lambda)$. ■

Note that if $\hat{\pi}$ denotes the shortest hyperpath w.r.t. the parametric weighting function, we have that $W_\lambda(\hat{\pi})$ is a lower bound on $f(\lambda)$ but $W(\hat{\pi}) = (W_1(\hat{\pi}), W_2(\hat{\pi}))$ is not necessarily

**Precondition:** Let $\left(W^{+}, W^{-}\right)$ define a search direction and let $\Phi$ be an ordered nondominated set of points. Given $W \in \Phi$ let $W^{next}$ denote the point following $W$ and let $W^{back}$ denote the point before $W$, if $W$ is the last element in $\Phi$ then $W^{next} := null$. Let increase denote a boolean.

**Initialization:** Use an SBT procedure to find the upper/right point $W^{\infty}$ and the lower/left point $W^{\varepsilon}$.

1 **if** $(W^{\infty} = W^{\varepsilon})$ **then** stop (there is only one nondominated solution)
2 **else** set $\Phi = \{W^{\varepsilon}, W^{\infty}\}$, $W^{+} = W^{\infty}$, $W^{-} = W^{\varepsilon}$
3 **while** $(W^{+} \neq W^{\varepsilon})$ **do** let $\lambda = \left|(W_2^{-} - W_2^{+})/(W_1^{-} - W_1^{+})\right|$, increase $= false$
4 **for** $(e \in \mathcal{E})$ **do** set $w_{\lambda}(e) = w_1(e)\,\lambda + w_2(e)$
5 find the minimal hyperpath $\check{\pi}$ w.r.t. the parametric weighting function
6 **call** procedure insert$(W(\check{\pi}), \Phi)$
7 **if** ($W(\check{\pi})$ nondominated and $W_1(\check{\pi}) \leq W_1^{+}$) **then** $W^{+} := W(\check{\pi})^{back}$
8 **if** ($W(\check{\pi})$ dominated or $W_1(\check{\pi}) > W_1^{-}$) **then** increase $=$ *true*
9 find the minimal hyperpath $\hat{\pi}$ w.r.t. upper bound Weighting function (4.7)
10 **if** $(\hat{\pi} \neq \check{\pi})$ **then call** procedure insert$(W(\hat{\pi}), \Phi)$
11 **if** ($W(\check{\pi})$ nondominated and $W_1(\check{\pi}) \leq W_1^{+}$) **then** $W^{+} := W(\check{\pi})^{back}$
12 **if** (increase and ($W(\hat{\pi})$ dominated or $W_1(\hat{\pi}) > W_1^{-}$)) **then** $W^{+} := W^{next}$
13 **end if**
14 set $W^{-} := W^{+next}$
15 **end while**
16 remove from $\Phi$ all nonextreme points

**Procedure 13:** Finding an approximation of the supported extreme nondominated points

a supported extreme nondominated point. In many cases, though, $W(\hat{\pi})$ is a supported extreme point, as we will see in Section 6.
Using the parametric weighting function gives $W_{\lambda}(\pi) \leq W_1(\pi)\,\lambda + W_2(\pi)$, nevertheless there exists another weighting function satisfying $\tilde{W}(\pi) = W_1(\pi)\,\lambda + W_2(\pi)$

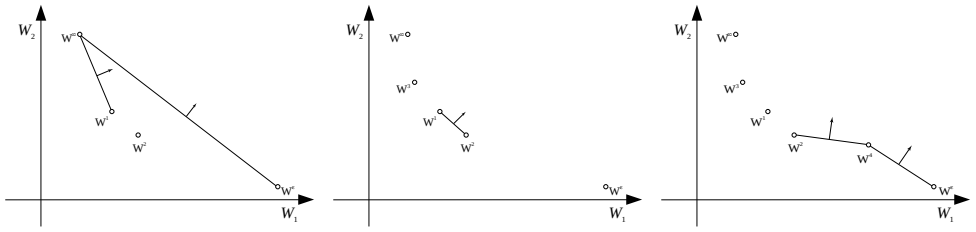
$$\tilde{W}(v) = \begin{cases} 0 & v = s \\ \max\limits_{u \in T(e_{\pi}(v))} \{W_1(v)\}\,\lambda + \max\limits_{u \in T(e_{\pi}(v))} \{W_2(v)\} + w_{\lambda}(e_{\pi}(v)) & v \in \mathcal{V}_{\pi} \setminus \{s\} \end{cases} \tag{4.7}$$

Here we have 3 labels in each node: $W_1$ and $W_2$ which are used to calculate $\tilde{W}$. If we solve SBT with Weighting function (4.7), we have to solve the following recursive equations

$$\tilde{W}(v) = \begin{cases} 0 & v = s \\ \min\limits_{e \in BS(v)} \left\{ \max\limits_{u \in T(e)} \{W_1(v)\}\,\lambda + \max\limits_{u \in T(e)} \{W_2(v)\} + w_{\lambda}(e) \right\} & v \in \mathcal{V} \setminus \{s\} \end{cases} \tag{4.8}$$

Solving the recursive equations (4.8) find a minimal hyperpath $\pi$ which often corresponds to a supported extreme nondominated point, but $\pi$ is not necessarily the hyperpath which solves (4.3). This can be seen in Figure 17 where hyperpath $\pi_1$ is the solution of equations (4.8), however, $\pi_2$ is the hyperpath which solves (4.3). Therefore $\tilde{W}(\pi)$ is an upper bound on $f(\lambda)$.
Sometimes a solution found using the upper bound Weighting function (4.7) cannot be found using the parametric weighting function. Therefore, by combining the two functions, we can find a better approximation of the frontier. Procedure 13 finds an approximation of the supported extreme nondominated points. The procedure is illustrated by the following example.

**Example 7** Assume that the initialization finds the points $W^{\infty}$ and $W^{\varepsilon}$ in Figure 18(a). First iteration now searches in the direction of the normal to the line defined by $W^{\infty}$ and $W^{\varepsilon}$. Suppose that $W^1$ is the point found on line 5. Because $W^1$ is nondominated and between $W^{+}$ and $W^{-}$ the conditions on lines 7 and 8 fail. Assuming that line 9 finds the same point, line 14 now sets $W^{-}$ to $W^1$. Second iteration now searches in the direction of

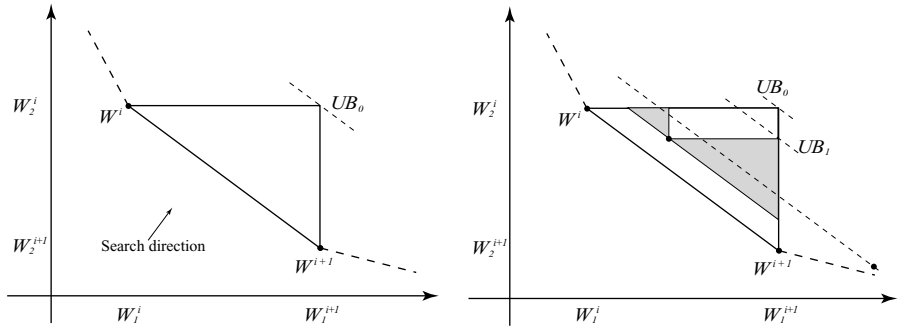


(a) Iteration 1 and 2 (b) Iteration 3 and 4 (c) Iteration 8 and 9

Figure 18: Finding supported extreme nondominated points with Procedure 13.

the normal to the line defined by $W^\infty$ and $W^1$. If we assume that line 5 finds point $W^1$, line 8 sets increase to *true* because $W^1$ is not a new point inside the triangle. Moreover, if line 9 finds point $W^2$, line 12 now sets $W^+ = W^1$ because both points found in the iteration are dominated or to the right of $W_1^-$. Iteration 3 now searches the triangle defined by $W^1$ and $W^2$. If line 5 now finds point $W^3$ in Figure 18(b), we have that line 7 set $W^+ = W^\infty$ because a new search direction is defined by the points $W^\infty$ and $W^3$. If we assume that line 9 does not find a new point, iteration 4 now searches in this direction. Assuming that no new points are found during iteration 4-6, iteration 7 now searches in the direction defined by $W^2$ and $W^\varepsilon$ and finds for instance the point $W^4$ in Figure 18(c). Therefore iteration 8 and 9 will search in the directions shown in Figure 18(c) and if no new points are found stop. Note that we sometimes search in the same direction more than once, but since the SBT procedure is so efficient this is now very costly. Moreover, at the end of Procedure 13, the ordered nondominated set $\Phi$ can contain nonextreme nondominated points. Therefore these are removed on line 16. ■

Procedure 13 finds a set of nondominated extreme points. This set is not necessarily equal to the set of supported extreme nondominated points in $\mathcal{W}_{Eff}$. Therefore it is possible that some points $W \in \Phi$ are dominated by a point in $\mathcal{W}_{Eff}$.




(a) Triangle $\Delta_i$ of $\Delta$ (b) Upper bound of $\Delta_i$.

Figure 19: Triangle search.

**Precondition:** Let $\Delta = \{\Delta_1, ..., \Delta_q\}$ denote the triangles of the supported extreme nondominated points and let $\Phi$ be an ordered nondominated set.

**Initialization:** Set $\Phi = \emptyset$

```
1  for (i = 1 to q) do consider triangle Δ_i and
2      set λ = |W_2^i − W_2^{i+1} / W_1^i − W_1^{i+1}|, Φ := Φ ∪ {W^i, W^{i+1}}
3      ub = W_1^{i+1}λ + W_2^i, lb = 0 and k = 1
4      for (e ∈ E) do set w_λ(e) = w_1(e) λ + w_2(e)
5      while (lb < ub) do use a k'th procedure and the parametric weighting function
6          to find the k'th best solution W_λ^k = W_1^k λ + W_2^k of H
7          set lb = W_λ^k
8          if (W_1^k < W_1^{i+1} and W_2^k < W_2^i) then (i.e. we are in the triangle)
9              call procedure insert(W^k, Φ).
10             if (W^k nondominated) then update ub
11         end if
12         k := k + 1
13     end while
14 end for
```

**Procedure 14:** Finding nondominated points (value/value).

### 4.2.2 Phase 2: Finding unsupported nondominated solutions

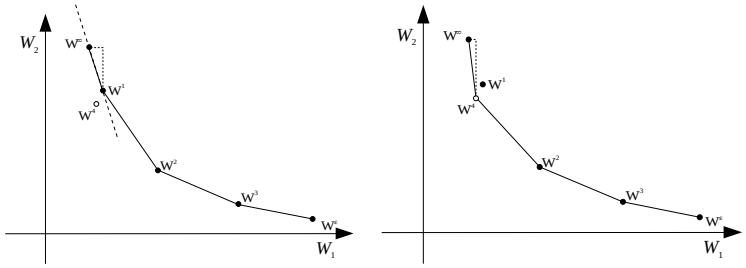
Assume that the first phase have been completed, i.e. an ordered nondominated set

$$\Phi = \left\{W^1, W^2, ..., W^{q+1}\right\}$$

of supported extreme nondominated points have been found. Let $\Delta = \{\Delta_1, ..., \Delta_q\}$ denote the triangles or gaps defined by $\Phi$ (see Figure 19(a)). Second phase searches each triangle using a $k'th$ best procedure until all unsupported nondominated points inside the triangle have been found. This is done by using modified weights $w_\lambda(e)$ on hypergraph $\mathcal{H}$ and hence the $k'th$ best procedure will search in the direction of the normal to the line between the two points which define the triangle (see Figure 19(a)). The procedure stops when an upper bound has been reached. At start the upper bound is $UB_0 = W_1^{i+1}\lambda + W_2^i$, however, when an new unsupported nondominated point is found in the triangle, we calculate an new upper bound (see Figure 19(b)). Furthermore, if an interactive approach is used the decision maker can set an upper bound where he is satisfied. Note that when we use a $k'th$ best procedure it is possible to find points outside the triangle (see Figure 19(b)). These points are not checked for dominance, instead the next $k'th$ solution is calculated. How the $k'th$ best procedure performs depends on the weighting functions considered and in which order the solutions are found. We again consider 2 choices of weighting function: Two value weighting functions and two distance weighting functions.

**Two value weighting functions** Here Procedure 12 on page 26 finds all supported extreme points of $\mathcal{W}_{Eff}$ and therefore, when we start the second phase we know how the triangles are defined. Moreover, a minimal hyperpath $\pi_\lambda$ which is a solution of (4.3) can be found using the parametric weighting function (4.4). We can now find all unsupported nondominated points using Procedure 14. Line 5 finds a new *k'th* hyperpath while $W_\lambda^k$ is below the upper bound. If $W^k$ is a nondominated point inside the triangle, we update the upper bound on line 10. Note when we use procedure insert to insert $W^k$ in $\Phi$ on line 9, we do not have to search $\Phi$ from start because we know $W_1^k > W_1^i$, i.e. we always start to search $\Phi$ from $W^i$ in procedure insert. The possible choices of the $k'th$ procedure on line 5 are: Procedure 7, i.e. we use the normal *k'th* procedure, Procedure 8 that is we branch on lower bounds, and finally Procedure 9 which is used if acyclic hypergraphs are considered.

**Two distance functions** When two distance weighting functions are considered, first phase (Procedure 13 on page 28) does not necessary find all supported extreme nondomi-




(a) Approximated frontier solutions and a new frontier solution

(b) New approximated frontier

Figure 20: Finding a better approximation during second phase.

nated points of $\mathcal{W}_{Eff}$ but just an approximation of $\mathcal{W}_{Eff}$. This only gives us an approximation of the triangles at the beginning of second phase. Assume that the approximation are the black points given in Figure 20(a). Now, if during the search of the triangle defined by $W^\infty$ and $W^1$, the point $W^4$ in Figure 20(a) is found, then $W^4$ is a new supported extreme nondominated point because the parametric weight of $W^4$ is below the parametric weight of $W^\infty$. I other words $W^4$ is below the dashed line between the points $W^\infty$ and $W^1$ in Figure 20(a)

Therefore a new approximation of the frontier is found and we search the new triangle defined by $W^\infty$ and $W^4$ (see Figure 20(b)). Let $\Phi$ be an ordered nondominated set containing all current nondominated points found by first and second phase and let $W^+$ denote the upper/left point of the triangle where the new supported extreme point is found. Then by removing all nonfrontier points above $W^+$ in $\Phi$, we remove a point such as $W^1$ in Figure 20(b). Moreover, the new triangle we have to search is defined by $W^+$ and the point following it in $\Phi$. The second-phase procedure for two distance functions is stated in Procedure 15. The boolean $newf$ on line 2 in Procedure 15 is set to true if a new frontier point is found. $W_{frontier}$ denotes the parametric weight of points on the line defined by $W^+$ and $W^-$. Therefore if $W_{now}$, the parametric weight of the current $k'th$ solution, is below $W_{frontier}$ (line 7), a new supported extreme nondominated point has been found. Line 8 now removes all the unsupported points above $W^+$, $newf$ is set to $true$ and we exit the while loop on line 10. Because $newf = true$ we now search for nondominated points in the triangle defined by $W^+$ and the new supported extreme nondominated point found. If no new frontier point is found during the inner while loop, we increase $W^+$ on line 17 and start searching the next triangle.

The bi-SBT problem is much harder to solve when considering two distance functions, as we will see in Section 6.This is due to the fact that one nondominated point often corresponds to many different hyperpaths. As a result the $k'th$ procedure used has to search more hyperpaths. Furthermore, since we use the parametric weighting function giving us a lower bound, many hyperpaths with an actual parametric weight over the upper bound is found. By using the upper bound function (4.7) instead of the parametric weighting function on line 4, we find hyperpaths with parametric weight equal to the actual parametric weight. However, since the parametric weight is an upper bound, we do not know whether all nondominated points are found when the procedure stops. Therefore using upper bound function (4.7) on line 4 instead of the parametric weighting function give us an approximation of the nondominated points. Nevertheless, computational testing in Section 6 shows that the approximation is good and much faster to calculate. The possible choices of the $k'th$ procedure on line 5 are like for two value weighting functions: Procedure 7, i.e. we use the normal $k$'$th$ procedure, Procedure 8 that is we branch on lower bounds, and finally Procedure 9 which is used if acyclic hypergraphs are considered.

**Precondition:** Let $\Phi = \{W_1, ..., W_q\}$ denote an ordered nondominated set containing the approximation of the supported extreme nondominated points found in phase one. Given $W \in \Phi$, let $W^{next}$ denote the point following $W$ and let $W^{next}$ be equal to $null$ if $W$ is the last point in $\Phi$.

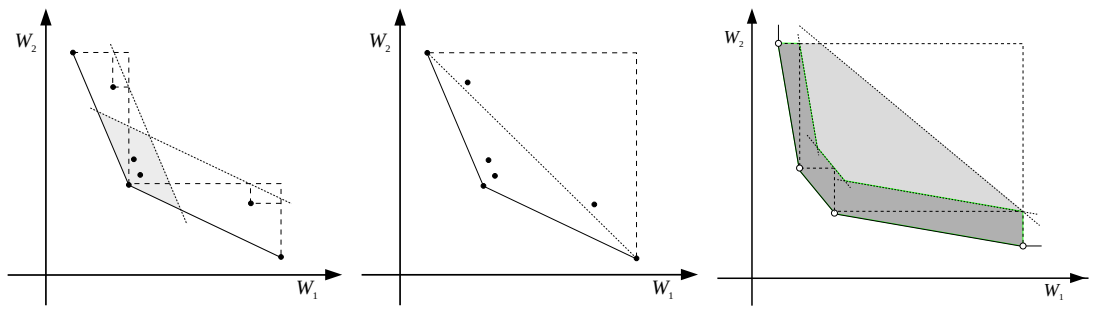
**Initialization:** $W^+ = W^1$

```
1  while (W^{+next} ≠ null) do W^- = W^{+next}, λ = |W_2^- − W_2^+ / W_1^- − W_1^+|
2     ub = W_1^+ λ + W_2^-, lb = 0, k = 1, W_frontier = W_1^+ λ + W_2^+, newf = false
3     for (e ∈ E) do set w_λ(e) = w_1(e) λ + w_2(e)
4     while (lb < ub) do use a k'th procedure and the parametric weighting function
5        to find the k'th solution W_λ^k of H
6        set LB = W_λ^k, W_now = W_1^k λ + W_2^k
7        if (W_now < W_frontier and W_1^+ ≤ W_1^k) then
8           remove all nonfrontier points from W^+ in Φ
9           set newf = true
10          break
11       end if
12       if (W_1^k < W_1^- and W_2^k < W_2^+) then call procedure insert(W^k, Φ).
13          if (W^k nondominated) then update ub
14       end if
15       k := k + 1
16    end while
17    if (newf = false) then W^+ = W^-
18 end while
```

**Procedure 15:** Finding nondominated points (distance/distance).

## 4.3 A more advanced $k'th$ diagonal search procedure

If we use a two-phases approach, we search each triangle with a $k'th$ best search procedure but when we use a *k'th* search procedure, it is also possible to find points outside the triangle. It is therefore possible to find the same points when different triangles are searched. This can be seen in Figure 21(a). Here we search each triangle, and the points in the shaded area is found twice. If instead, we use a diagonal search for the whole area, i.e. we search in the direction of the normal to the line between the upper/left point and the lower/right point in Figure 21(b), we only find these points once.
This has to be compared with the fact that the *k'th* diagonal procedure sometimes has to search a larger area than the two-phases method. Figure 21(c) shows such a case where the second phase procedure in the worst case searches the dark shaded area, but the *k'th* diagonal procedure in the worst case has to search the light shaded area before it stops. Note




(a) Overlappeing with two-phases search

(b) Diagonal search

(c) Disadvantages of the diagonal search

Figure 21: Two phase search and diagonal search.

**Precondition:** Let $\Phi$ denote an ordered nondominated ordered set.

**Initialization:** Use an SBT procedure to find the upper/right point $W^{\infty}$ and the lower/left point $W^{\varepsilon}$. Set $\lambda = |W_2^{\infty} - W_2^{\varepsilon}/W_1^{\varepsilon} - W_1^{\infty}|$, $ub = W_1^{\varepsilon}\lambda + W_2^{\infty}$, $\Phi = \emptyset$, $k = 1$

1 **if** ($W^{\infty} = W^{\varepsilon}$) **then** stop (there is only one nondominated solution)
2 **for** ($e \in \mathcal{E}$) **do** set $w_{\lambda}(e) = w_1(e)\,\lambda + w_2(e)$
3 **while** ($lb < ub$) **do** use a *k'th* procedure to find the *k'th* solution $W_{\lambda}^{k}$ of $\mathcal{H}$
4 corresponding to point $W^k$
5 set $lb = W_{\lambda}^{k}$
6 **call** procedure insert($W^k, \Phi$).
7 **if** ($W^k$ nondominated) **then** update $ub$
8 set $k := k + 1$
9 **end while**

**Procedure 16:** A *k'th* diagonal search procedure.

that even though the second phase procedure searches an smaller area, some points in this area will be found more than once. This cost has to be compared with the cost of a larger area for the *k'th* diagonal procedure which is stated in Procedure 16. If we consider two distance weighting functions, we can, like before, use the upper bound weighting function or the parametric weighting function. Moreover, we have the same possible choices of $k'th$ procedure on line 3 as in the two-phases procedure.

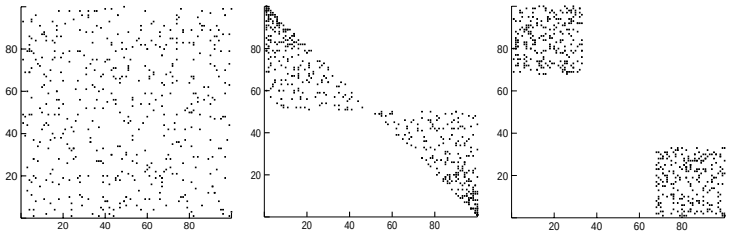
# 5 Random Hypergraphs

To test the procedures in Section 4, a random hypergraph generator (B-MAKER) was made. B-MAKER generates both general hypergraphs and acyclic hypergraphs. Due to the fact that a hypergraph is a much more complex structure than a digraph, B-MAKER needs the input parameters given in Table 1.

| | | | |
|---|---|---|---|
| $n$ | $m_a$ | $hsize_{\min}$ | $hsize_{\max}$ |
| $m_h$ | $size_{FS(s)}$ | $size_{BS(t)}^{arc}$ | $size_{BS(t)}^{harc}$ |
| $ubw_{arc}^{FS(s)}$ | $lbw_{arc}^{FS(s)}$ | $lbw_{arc}^{other}$ | $ubw_{arc}^{other}$ |
| $lbw_{harc}$ | $ubw_{harc}$ | *weight-type* | *seed* |

Table 1: Input parameters for B-MAKER.

Here $n$, $m_a$ and $m_h$ are the number of nodes, arcs and true hyperarcs, respectively. The size of each true hyperarc $e$ belongs to $[hsize_{\min}, hsize_{\max}]$ and the weights assigned to $e$ belongs to $[lbw_{harc}, ubw_{harc}]$. Node number one is the source node $s$ with $BS(s) = \emptyset$. The number of arcs in the forward star of $s$ are $size_{FS(s)}$ and the weights of each arc belong to $\left[lbw_{arc}^{FS(s)}, ubw_{arc}^{FS(s)}\right]$. There are no hyperarcs in the forward star of $s$. All other arcs generated have weights from $lbw_{arc}$ to $ubw_{arc}$. The destination node $t$ is node $n$. Here $FS(t) = \emptyset$ and the number of arcs and hyperarcs in $BS(t)$ are $size_{BS(t)}^{arc}$ and $size_{BS(t)}^{harc}$, respectively. All random numbers were generated with an initial input number *seed*. Assume a hyperarc $e$ is considered with weights $w_i \in [lbw, ubw]$, $i = 1, 2$. The correlation between



(a) Points generated with weight-type 0.

(b) Points generated with weight-type 2.

(c) Points generated with weight-type 4.

Figure 22: Points generated in $[1, 100] \times [1, 100]$ for different weight-types.

the weights $(w_1, w_2)$ now depends on the number *weight-type* specified in the input file.

$$
\begin{aligned}
weight - type = 0 &: \text{Both weights are random in } [lbw, ubw]. \\
weight - type = 2 &: w_1 < \frac{(ubw + lbw)}{2} \Rightarrow w_2 \in [ubw - (w_1 - lbw), ubw] \\
&\quad\; w_1 \geq \frac{(ubw + lbw)}{2} \Rightarrow w_2 \in [lbw, lb + (ubw - w_1)] \\
weight - type = 4 &: \text{"...if one weight is between 1 and 33 the other is between 67 and 100"} \\
&\quad\; \text{Skriver and Andersen [26].}
\end{aligned}
$$

Weights generated with option 0,2 and 4 with $[lbw, ubw] = [1, 100]$ can be seen in Figure 22. Weight-type 2 and 4 generate negatively correlated weights. The way B-MAKER generates arcs and hyperarcs depends on whether general or acyclic hypergraphs are considered:

1. **General case:** First, randomly generate the arcs and hyperarcs in $FS(s)$ and $BS(t)$ specified. The arc $(s, t)$ is not generated. Secondly, randomly generate hyperarcs for node $2, ..., n-1$. No repeated arcs are generated.

2. **Acyclic case:** First, an arc from node $s$ to node $2, ..., size_{FS(s)} + 1$ is made. No other arcs and hyperarcs enter these nodes. Second, all other hyperarcs are randomly generated satisfying that the head node number is bigger than the tail node numbers, i.e. a valid ordering of $\mathcal{H}$ is $V = (1, ..., n)$. No repeated arcs are generated.

# 6 Computational Results

In this section we test the procedures described in Section 4. The procedures have all been implemented in C++ and tested on a Hewlett Packard 9000/720 series computer with 112MB RAM, which is a small and slow computer[6].

Each general hypergraph is represented using a forward and backward star representation of pointers and numbers. Only a backward star representation is used for acyclic hypergraphs. The SBT procedure (Procedure 3 on page 9) is implemented using a heap to store the candidate set and an element is picked using Dijkstra's principle, i.e. we pick the element with lowest weight. The subhypergraphs in the *k'th* procedure are represented using a dynamic tree together with a heap of pointers. For more details on implementation, see Appendix A.

| Hgraph | Nodes | Arcs | Harcs | $FS_{arc}(s)$ | $BS_{arc}(t)$ | $BS_{harc}(t)$ | Hgraph | Nodes | Arcs | Harcs | $FS_{arc}(s)$ | $BS_{arc}(t)$ | $BS_{harc}(t)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 100 | 300 | 6000 | 10 | 10 | 50 | 31 | 1000 | 1400 | 4000 | 5 | 5 | 20 |
| 2 | | 400 | 8000 | 10 | 10 | 50 | 32 | | 1500 | 5000 | 5 | 5 | 20 |
| 3 | | 500 | 10000 | 10 | 10 | 50 | 33 | | 1600 | 7000 | 5 | 5 | 20 |
| 4 | 200 | 600 | 12000 | 20 | 20 | 100 | 34 | 2000 | 2800 | 8000 | 10 | 10 | 40 |
| 5 | | 800 | 16000 | 20 | 20 | 100 | 35 | | 3000 | 10000 | 10 | 10 | 40 |
| 6 | | 1000 | 20000 | 20 | 20 | 100 | 36 | | 3200 | 14000 | 10 | 10 | 40 |
| 7 | 300 | 900 | 18000 | 30 | 30 | 150 | 37 | 3000 | 4200 | 12000 | 15 | 15 | 60 |
| 8 | | 1200 | 24000 | 30 | 30 | 150 | 38 | | 4500 | 15000 | 15 | 15 | 60 |
| 9 | | 1500 | 30000 | 30 | 30 | 150 | 39 | | 4800 | 21000 | 15 | 15 | 60 |
| 10 | 400 | 1200 | 24000 | 40 | 40 | 200 | 40 | 4000 | 5600 | 16000 | 20 | 20 | 80 |
| 11 | | 1600 | 32000 | 40 | 40 | 200 | 41 | | 6000 | 20000 | 20 | 20 | 80 |
| 12 | | 2000 | 40000 | 40 | 40 | 200 | 42 | | 6400 | 28000 | 20 | 20 | 80 |
| 13 | 500 | 1500 | 30000 | 50 | 50 | 250 | 43 | 5000 | 7000 | 20000 | 25 | 25 | 100 |
| 14 | | 2000 | 40000 | 50 | 50 | 250 | 44 | | 7500 | 25000 | 25 | 25 | 100 |
| 15 | | 2500 | 50000 | 50 | 50 | 250 | 45 | | 8000 | 35000 | 25 | 25 | 100 |
| 16 | 600 | 1800 | 36000 | 60 | 60 | 300 | 46 | 6000 | 8400 | 24000 | 30 | 30 | 120 |
| 17 | | 2400 | 48000 | 60 | 60 | 300 | 47 | | 9000 | 30000 | 30 | 30 | 120 |
| 18 | | 3000 | 60000 | 60 | 60 | 300 | 48 | | 9600 | 42000 | 30 | 30 | 120 |
| 19 | 700 | 2100 | 42000 | 70 | 70 | 350 | 49 | 7000 | 9800 | 28000 | 35 | 35 | 140 |
| 20 | | 2800 | 56000 | 70 | 70 | 350 | 50 | | 10500 | 35000 | 35 | 35 | 140 |
| 21 | | 3500 | 70000 | 70 | 70 | 350 | 51 | | 11200 | 49000 | 35 | 35 | 140 |
| 22 | 800 | 2400 | 48000 | 80 | 80 | 400 | 52 | 8000 | 11200 | 32000 | 40 | 40 | 160 |
| 23 | | 3200 | 64000 | 80 | 80 | 400 | 53 | | 12000 | 40000 | 40 | 40 | 160 |
| 24 | | 4000 | 80000 | 80 | 80 | 400 | 54 | | 12800 | 56000 | 40 | 40 | 160 |
| 25 | 900 | 2700 | 54000 | 90 | 90 | 450 | 55 | 9000 | 12600 | 36000 | 45 | 45 | 180 |
| 26 | | 3600 | 72000 | 90 | 90 | 450 | 56 | | 13500 | 45000 | 45 | 45 | 180 |
| 27 | | 4500 | 90000 | 90 | 90 | 450 | 57 | | 14400 | 63000 | 45 | 45 | 180 |
| 28 | 1000 | 3000 | 60000 | 100 | 100 | 500 | 58 | 10000 | 14000 | 40000 | 50 | 50 | 200 |
| 29 | | 4000 | 80000 | 100 | 100 | 500 | 59 | | 15000 | 50000 | 50 | 50 | 200 |
| 30 | | 5000 | 100000 | 100 | 100 | 500 | 60 | | 16000 | 70000 | 50 | 50 | 200 |

Table 2: Benchmark problems generated with B-MAKER with a hyperarc size between 3 and 5.

## 6.1 Benchmark problems

To check the efficiency of the procedures 60 random hypergraphs where generated with B-MAKER and tested. Specifications of the hypergraphs is shown in Table 2. The hyperarc size in hypergraphs 1-60 are between 3 and 5. The weights for each arc is between 1 and 1000, and for each true hyperarc between 1 and 100. This favor hyperpaths with true hyperarcs. The generated hypergraphs are divided into two groups: Hypergraphs 1-30 are dense and the average number of hyperarcs in the backward star of a node are between 63 and 105. Hypergraphs 31-60 are sparse and the average number of hyperarcs in the backward star of a node are between 4.4 and 8.6. The number of arcs in $FS\,(s)$ have been set to 10% of the number of nodes for the dense hypergraphs, and to 0.5% for the sparse hypergraphs. The same are valid for the number of arcs in $BS\,(t)$. The number of true hyperarcs in $BS\,(t)$ have been set to 50% of the number of nodes for the dense hypergraphs, and to 2% for the

[6]The slowest at the department. The fastest performs approximately 10 times faster.

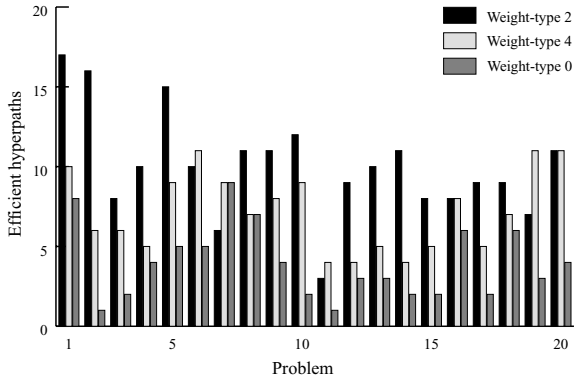



Figure 23: The number of efficient hyperpaths for 20 selected hypergraphs with weighttype 0,2 and 4. Two sum functions are considered.

sparse hypergraphs. For each problem in Table 2 a general and an acyclic hypergraph were generated with B-MAKER.

## 6.2 General hypergraphs

To compare the number of efficient hyperpaths when the hyperarc weights are correlated differently, each hypergraph in Table 2 were generated with weight-type option 0,2 and 4. The number of nondominated points for 20 selected hypergraphs are shown in Figure 23. We see, that the number of nondominated points in most cases are highest when using 2 as weight-type and lowest when the weights are uncorrected (weight-type option 0). We therefore only use weight-type option 2 to compare the procedures in the following.

### 6.2.1 Two sum functions

Because the sum function (all multipliers equal 1) satisfies Dijkstra's principle, we use this function to compare the value procedures for general hypergraphs. We compare the following procedures:

***k'th* simple** We here use the simple search procedure presented in Section 4.1, i.e. use a *k'th* procedure in weight ones direction (Procedure 7).

***k'th* diagonal** Use the diagonal search procedure presented in Section 4.3 together with the *k'th* procedure which find the $k$ best solutions in the right order (Procedure 7).

***k'th* diagonal (lb)** Equal to *k'th* diagonal, but use the *k'th* lower bound procedure instead which finds all solutions below an upper bound (Procedure 8).

**Two-phases** We here consider the two-phases procedures in Section 4.2. First, we use the phase one procedure (Procedure 12) in Section 4.2.1. Next, we use the second phase procedure (Procedure 14) in Section 4.2.2 together with the *k'th* procedure (Procedure 7).

**Two-phases (lb)** Equal to two-phases, but second phase use the *k'th* lower bound procedure instead (Procedure 8).

The results for hypergraphs 1-30 are shown in Table 3. The CPU time is reported in seconds, and column 'Hpaths' contains the number of *k'th* best solutions picked in each procedure. This number is the same in the lower bound procedures and therefore only reported once. Similar '1. phase' is the same for 'Phase two' and 'Two (lb)'. 'BSize' denote the average

| | $k'th$ simple | | $k'th$ diagonal | | (lb) | 1. phase | | Phase two | | | | | Two (lb) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hgraph | Hpaths | CPU | Hpaths | CPU | CPU | Extreme | CPU | Triangles | Hpaths | CPU | Bsize | CPU total | CPU | CPU total | Nondom. |
| 1 | 40 | 12,38 | 5 | 0,74 | 0,45 | 3 | 0,31 | 2 | 8 | 0,92 | 2,89 | 1,23 | 0,39 | 0,70 | 3 |
| 2 | $500^{1}$ | 273,07 | $300^{13}$ | 110,54 | 29,10 | 6 | 1,05 | 5 | 53 | 17,28 | 4,38 | 18,33 | 4,79 | 5,84 | 13 |
| 3 | $500^{7}$ | 316,01 | $300^{16}$ | 123,21 | 33,79 | 5 | 1,09 | 4 | 55 | 22,75 | 4,13 | 23,84 | 5,85 | 6,94 | 17 |
| 4 | $500^{5}$ | 361,12 | 15 | 8,74 | 2,56 | 3 | 0,81 | 2 | 18 | 10,08 | 5,32 | 10,89 | 2,49 | 3,30 | 9 |
| 5 | $500^{1}$ | 591,24 | 170 | 192,05 | 35,65 | 6 | 2,42 | 5 | 31 | 24,49 | 6,07 | 26,91 | 5,57 | 7,99 | 7 |
| 6 | $500^{3}$ | 600,79 | 124 | 143,27 | 31,90 | 7 | 3,56 | 6 | 54 | 50,20 | 4,12 | 53,76 | 12,96 | 16,52 | 16 |
| 7 | $500^{4}$ | 798,46 | $300^{10}$ | 345,49 | 69,91 | 8 | 3,76 | 7 | 28 | 21,95 | 5,42 | 25,71 | 5,31 | 9,07 | 10 |
| 8 | $500^{5}$ | 1248,73 | $300^{8}$ | 390,53 | 97,15 | 6 | 3,82 | 5 | 30 | 30,96 | 4,51 | 34,78 | 8,48 | 12,30 | 10 |
| 9 | $500^{3}$ | 1535,85 | 20 | 23,86 | 8,72 | 5 | 3,81 | 4 | 19 | 18,99 | 4,03 | 22,80 | 6,20 | 10,01 | 8 |
| 10 | $500^{4}$ | 1154,76 | 57 | 77,43 | 19,24 | 5 | 3,14 | 4 | 32 | 31,77 | 4,90 | 34,91 | 9,55 | 12,69 | 11 |
| 11 | $500^{2}$ | 1776,55 | 90 | 157,14 | 39,61 | 5 | 4,17 | 4 | 28 | 38,62 | 4,26 | 42,79 | 10,71 | 14,88 | 10 |
| 12 | $500^{8}$ | 2167,48 | 49 | 85,74 | 27,25 | 6 | 6,34 | 5 | 33 | 55,53 | 3,97 | 61,87 | 15,48 | 21,82 | 10 |
| 13 | $500^{5}$ | 1634,35 | 114 | 173,25 | 47,32 | 5 | 3,96 | 4 | 30 | 35,99 | 4,15 | 39,95 | 11,01 | 14,97 | 8 |
| 14 | $500^{5}$ | 1964,82 | $300^{10}$ | 614,76 | 164,98 | 7 | 7,67 | 6 | 39 | 63,56 | 4,03 | 71,23 | 18,79 | 26,46 | 11 |
| 15 | $500^{7}$ | 2446,89 | $300^{13}$ | 747,64 | 207,16 | 6 | 8,13 | 5 | 35 | 78,55 | 4,61 | 86,68 | 21,28 | 29,41 | 15 |
| 16 | $500^{4}$ | 2102,92 | 15 | 15,95 | 8,58 | 4 | 3,74 | 3 | 25 | 22,07 | 2,94 | 25,81 | 11,29 | 15,03 | 8 |
| 17 | $500^{5}$ | 2626,52 | 71 | 169,33 | 48,58 | 5 | 6,43 | 4 | 22 | 31,61 | 4,13 | 38,04 | 12,54 | 18,97 | 7 |
| 18 | $500^{3}$ | 3147,12 | 59 | 158,70 | 50,55 | 5 | 8,01 | 4 | 28 | 59,90 | 3,80 | 67,91 | 20,55 | 28,56 | 10 |
| 19 | $500^{4}$ | 2200,73 | 258 | 789,56 | 156,46 | 6 | 7,12 | 5 | 32 | 50,35 | 5,26 | 57,47 | 16,90 | 24,02 | 9 |
| 20 | $500^{5}$ | 2885,50 | 26 | 57,98 | 22,63 | 4 | 6,02 | 3 | 24 | 36,66 | 3,05 | 42,68 | 17,35 | 23,37 | 8 |
| 21 | $500^{4}$ | 3763,29 | 17 | 42,21 | 19,13 | 3 | 5,33 | 2 | 19 | 41,15 | 2,60 | 46,48 | 17,31 | 22,64 | 6 |
| 22 | $500^{6}$ | 2275,59 | 26 | 50,12 | 19,23 | 3 | 3,64 | 2 | 14 | 18,06 | 5,20 | 21,70 | 8,44 | 12,08 | 9 |
| 23 | $500^{3}$ | 4173,99 | 25 | 67,75 | 24,69 | 3 | 4,87 | 2 | 21 | 49,17 | 4,11 | 54,04 | 17,65 | 22,52 | 5 |
| 24 | $500^{4}$ | 5045,85 | 269 | 1179,54 | 335,93 | 6 | 14,61 | 5 | 34 | 91,80 | 3,53 | 106,41 | 37,14 | 51,75 | 11 |
| 25 | $500^{5}$ | 3132,78 | 14 | 21,61 | 12,57 | 2 | 2,47 | 1 | 14 | 20,05 | 2,63 | 22,52 | 10,24 | 12,71 | 7 |
| 26 | $500^{6}$ | 3772,95 | 15 | 31,51 | 17,85 | 3 | 5,55 | 2 | 28 | 54,93 | 2,59 | 60,48 | 27,36 | 32,91 | 12 |
| 27 | $500^{5}$ | 5344,87 | $300^{10}$ | 1488,84 | 398,54 | 7 | 18,46 | 6 | 40 | 142,22 | 4,33 | 160,68 | 46,55 | 65,01 | 11 |
| 28 | $500^{6}$ | 3356,62 | 22 | 46,99 | 21,06 | 6 | 10,26 | 5 | 33 | 54,30 | 3,72 | 64,56 | 25,38 | 35,64 | 11 |
| 29 | $500^{4}$ | 4851,12 | $300^{8}$ | 1653,86 | 349,25 | 3 | 7,44 | 2 | 13 | 35,34 | 6,47 | 42,78 | 13,21 | 20,65 | 8 |
| 30 | $500^{3}$ | 5617,55 | 35 | 137,43 | 53,44 | 5 | 13,89 | 4 | 42 | 115,38 | 3,02 | 129,27 | 55,71 | 69,60 | 12 |

Table 3: Results when testing on general dense hypergraphs with two sum functions. CPU times are reported in seconds.

number of hyperarcs we branch on, i.e. the average number of subhypergraphs created when a *k'th* best solution is picked. The number of extreme supported nondominated points found by the first phase procedure are shown in column 'Extreme', and the last column contains the total number of nondominated points. The *k'th* simple and *k'th* diagonal procedures were set to automatically terminate when 500 and 300 hyperpaths had been picked, respectively. If this were the case the number of nondominated points found is shown as a small number raised in the 'Hpaths' column.

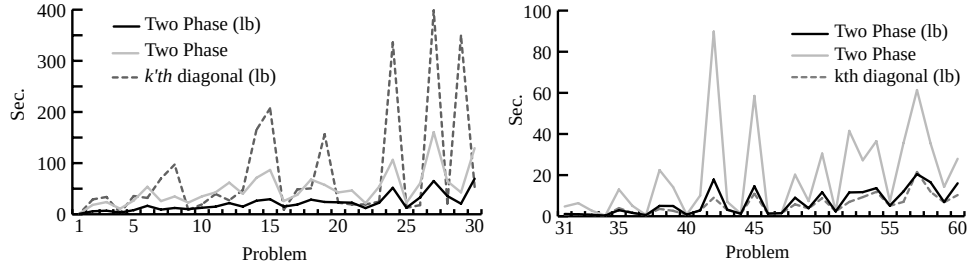
It is clear from the results in Table 3 that the *k'th* simple procedure is very slow compared to the other procedures. The procedure terminates in all hypergraphs when it has picked 500 hyperpaths, except for hypergraph one. The *k'th* diagonal search procedure also terminates, before all nondominated points are found in some cases. However, in hypergraphs where both the *k'th* simple and the *k'th* diagonal procedure terminates before all nondominated

| | $k'th$ simple | | $k'th$ diag. | | (lb) | 1. phase | | Phase two | | | | | Two (lb) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hgraph | Hpaths | CPU | Hpaths | CPU | CPU | Extreme | CPU | Triangles | Hpaths | CPU | Bsize | CPU total | CPU | CPU total | Nondom. |
| 31 | 382 | 264,35 | 10 | 4,83 | 0,86 | 4 | 0,48 | 3 | 11 | 4,26 | 19,27 | 4,74 | 0,63 | 1,11 | 5 |
| 32 | 53 | 51,26 | 9 | 6,37 | 0,91 | 2 | 0,25 | 1 | 9 | 6,12 | 17,39 | 6,37 | 0,71 | 0,96 | 4 |
| 33 | 2 | 1,00 | 4 | 2,55 | 0,68 | 2 | 0,34 | 1 | 4 | 2,36 | 14,16 | 2,70 | 0,36 | 0,70 | 2 |
| 34 | 1 | 0,33 | 1 | 0,34 | 0,33 | 1 | 0,34 | 0 | 0 | 0,00 | 0,00 | 0,34 | 0,00 | 0,34 | 1 |
| 35 | 66 | 104,42 | 18 | 23,79 | 3,97 | 4 | 1,42 | 3 | 10 | 11,60 | 14,61 | 13,02 | 1,53 | 2,95 | 4 |
| 36 | 14 | 29,27 | 4 | 4,74 | 1,60 | 2 | 0,82 | 1 | 4 | 4,23 | 11,88 | 5,05 | 0,84 | 1,66 | 3 |
| 37 | 1 | 0,54 | 1 | 0,56 | 0,56 | 1 | 0,56 | 0 | 0 | 0,00 | 0,00 | 0,56 | 0,00 | 0,56 | 1 |
| 38 | 17 | 33,61 | 9 | 17,54 | 3,57 | 3 | 1,68 | 2 | 12 | 20,77 | 15,92 | 22,45 | 3,35 | 5,03 | 5 |
| 39 | 13 | 35,08 | 4 | 6,82 | 2,63 | 3 | 2,24 | 2 | 8 | 11,95 | 12,67 | 14,19 | 2,74 | 4,98 | 3 |
| 40 | 1 | 0,77 | 1 | 0,79 | 0,78 | 1 | 0,78 | 0 | 0 | 0,00 | 0,00 | 0,78 | 0,00 | 0,78 | 1 |
| 41 | 4 | 10,96 | 4 | 9,54 | 2,81 | 2 | 1,41 | 1 | 4 | 8,68 | 18,96 | 10,09 | 1,46 | 2,87 | 2 |
| 42 | $500^4$ | 3042,86 | 13 | 54,04 | 9,00 | 5 | 5,55 | 4 | 24 | 84,25 | 14,81 | 89,80 | 12,30 | 17,85 | 5 |
| 43 | 10 | 17,31 | 4 | 6,40 | 3,08 | 2 | 1,58 | 1 | 4 | 5,42 | 16,65 | 7,00 | 1,61 | 3,19 | 3 |
| 44 | 1 | 1,18 | 1 | 1,21 | 1,23 | 1 | 1,22 | 0 | 0 | 0,00 | 0,00 | 1,22 | 0,00 | 1,22 | 1 |
| 45 | 76 | 520,65 | 12 | 60,18 | 10,96 | 4 | 5,65 | 3 | 14 | 52,86 | 12,82 | 58,51 | 8,93 | 14,58 | 5 |
| 46 | 1 | 1,24 | 1 | 1,29 | 1,27 | 1 | 1,27 | 0 | 0 | 0,00 | 0,00 | 1,27 | 0,00 | 1,27 | 1 |
| 47 | 1 | 1,51 | 1 | 1,50 | 1,51 | 1 | 1,50 | 0 | 0 | 0,00 | 0,00 | 1,50 | 0,00 | 1,50 | 1 |
| 48 | 10 | 46,65 | 4 | 13,04 | 5,78 | 3 | 4,91 | 2 | 6 | 15,30 | 12,94 | 20,21 | 4,10 | 9,01 | 3 |
| 49 | 2 | 3,97 | 3 | 6,53 | 3,80 | 2 | 2,29 | 1 | 3 | 5,08 | 16,70 | 7,37 | 1,63 | 3,92 | 2 |
| 50 | 18 | 96,49 | 8 | 26,33 | 8,77 | 3 | 4,46 | 2 | 10 | 26,02 | 14,52 | 30,48 | 7,21 | 11,67 | 5 |
| 51 | 1 | 2,29 | 1 | 2,35 | 2,35 | 1 | 2,35 | 0 | 0 | 0,00 | 0,00 | 2,35 | 0,00 | 2,35 | 1 |
| 52 | 14 | 76,87 | 6 | 22,55 | 6,97 | 3 | 4,40 | 2 | 10 | 37,11 | 22,91 | 41,51 | 7,17 | 11,57 | 5 |
| 53 | 230 | 1703,87 | 7 | 26,56 | 9,30 | 3 | 5,25 | 2 | 8 | 21,97 | 13,75 | 27,22 | 6,47 | 11,72 | 3 |
| 54 | $500^2$ | 5638,32 | 7 | 38,38 | 11,86 | 3 | 6,72 | 2 | 7 | 29,80 | 12,45 | 36,52 | 6,94 | 13,66 | 3 |
| 55 | 2 | 3,78 | 3 | 6,12 | 5,06 | 2 | 3,04 | 1 | 3 | 4,18 | 8,14 | 7,22 | 2,16 | 5,20 | 2 |
| 56 | 103 | 874,60 | 4 | 19,18 | 7,00 | 3 | 5,92 | 2 | 7 | 29,61 | 14,92 | 35,53 | 6,13 | 12,05 | 3 |
| 57 | 33 | 319,98 | 12 | 79,01 | 21,56 | 3 | 8,07 | 2 | 10 | 53,30 | 11,00 | 61,37 | 12,44 | 20,51 | 4 |
| 58 | 29 | 188,18 | 8 | 33,30 | 11,91 | 4 | 7,91 | 3 | 10 | 27,50 | 21,03 | 35,41 | 8,61 | 16,52 | 4 |
| 59 | 2 | 7,10 | 3 | 12,78 | 6,77 | 2 | 4,06 | 1 | 3 | 10,20 | 14,91 | 14,26 | 2,87 | 6,93 | 2 |
| 60 | 30 | 261,60 | 4 | 18,17 | 10,30 | 3 | 8,73 | 2 | 6 | 19,21 | 8,40 | 27,94 | 7,29 | 16,02 | 3 |

Table 4: Results when testing on general sparse hypergraphs with two sum functions. CPU times are reported in seconds.

points are found, the *k'th* diagonal procedure finds considerable more nondominated points. For instance, the *k'th* simple procedure finds only one nondominated point in hypergraph two, but the *k'th* diagonal procedure finds thirteen.

If we compare the lower bound procedures with the normal ones, we see that branching on lower bounds gives a faster computation time, than if we branch in the right order. This is due to the fact that when branching on lower bounds, we only calculate a shortest hyperpath when a subhypergraph with minimal lower bound is picked. Moreover, this lower bound is virtually always equal the true minimal weight of the shortest hyperpath (in more than 95% of the cases). If we consider the two-phases (lb) procedure in hypergraph 30, for instance, we calculated 42 hyperpaths and all other computations are made using the simple lower bound. On the other hand, if we branch in the right order, we have to calculate the shortest hyperpath for each subhypergraph generated when we pick a shortest hyperpath, i.e. we




(a) Dense hypergraphs.

(b) Sparse hypergraphs.

Figure 24: CPU times for general hypergraphs using the sum function.

approximately calculate 42 times 3,02 hyperpaths in hypergraph 30 before the second phase terminate. Therefore branching on lower bounds are much faster.
If we consider the two-phases procedures and compare the CPU time for the first phase with the second phase, we see that it is relatively fast to compute the extreme supported nondominated points, the triangle search is more time consuming.
The CPU times for procedure *k'th* diagonal (lb), two-phases and two-phases (lb) are shown in Figure 24(a). It is easy to see that the two-phases (lb) procedure outperform the other procedures. In a few cases the *k'th* diagonal (lb) procedure is actually better than the two-phases (lb) procedure, but the procedure is not very stable. Often it stops before all nondominated points are found, and even the two-phases procedure outperforms the *k'th* diagonal (lb) procedure in most cases.
If we consider the sparse hypergraphs (31-60) instead the conclusion is not so clear. The results for hypergraphs 31-60 are shown in Table 4. Here the number of nondominated points are low; some hypergraphs in fact only have one efficient hyperpath. If the CPU times in Figure 24(b) are considered, we see that the *k'th* diagonal (lb) procedure performs best in most cases. This is due to the fact, that when the hypergraph is sparse the number of nondominated points inside the area, the *k'th* diagonal (lb) procedure in the worst case has to search, is very low (see Figure 21(c)). The two-phases (lb) procedure on the other hand, first has to find extreme supported nondominated points, and second search the triangles which imply finding some points twice. Therefore often the two-phases (lb) procedure has to calculate more hyperpaths in total than the *k'th* diagonal (lb) procedure resulting in longer computation times.

### 6.2.2 Two distance functions

We consider two distance functions and since branching on lower bounds are fastest we compare the following procedures:

***k'th* diagonal (lb)** Use the diagonal search procedure presented in Section 4.3 together with the *k'th* lower bound procedure which finds all solutions below an upper bound (Procedure 8).

**Two-phases (lb)** The two-phases procedures in Section 4.2. First, we use the phase one procedure (Procedure 12) in Section 4.2.1. Next, we use the second phase procedure (Procedure 14) in Section 4.2.2, together with the *k'th* lower bound procedure.

**Two-phases (ub)** Instead of using the parametric weighting function in second phase, we use the upper bound weighting function (4.7). This only gives us an approximation of the set of nondominated points.

| | $k'th$ diag. (lb) | | 1. Phase | | | Phase Two (lb) | | | | | | Phase Two (ub) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hgraph | HPaths | CPU | Extreme | Nonfrontier | CPU | Triangles | HPaths | CPU | CPU total | Unfinished | New extreme | HPaths | CPU | CPU total | Unfinished | Nondom. |
| 1 | $500^{6}$ | 36,1 | 11 | 5 | 5,5 | 11 | $5500^{26}$ | 406,2 | 411,7 | 11 | 1 | $1679^{42}$ | 126,6 | 132,0 | 3 | ? |
| 2 | $500^{6}$ | 52,1 | 11 | 9 | 13,2 | 11 | $5500^{30}$ | 611,2 | 624,4 | 11 | 2 | $3622^{43}$ | 398,9 | 412,1 | 7 | ? |
| 3 | $500^{12}$ | 65,5 | 15 | 5 | 15,0 | 14 | $7000^{33}$ | 1002,9 | 1017,9 | 14 | 2 | $2340^{39}$ | 323,2 | 338,2 | 3 | ? |
| 4 | $500^{5}$ | 82,8 | 12 | 2 | 13,8 | 10 | $5000^{18}$ | 823,7 | 837,6 | 10 | 1 | $1680^{28}$ | 278,1 | 291,9 | 3 | ? |
| 5 | $500^{6}$ | 113,9 | 6 | 2 | 9,6 | 5 | $2500^{13}$ | 569,7 | 579,3 | 5 | 0 | $1173^{24}$ | 273,0 | 282,6 | 2 | ? |
| 6 | $500^{7}$ | 138,0 | 9 | 6 | 20,2 | 8 | $4000^{26}$ | 1131,4 | 1151,6 | 8 | 0 | $914^{37}$ | 262,8 | 283,0 | 0 | ? |
| 7 | $500^{6}$ | 137,5 | 8 | 6 | 22,0 | 7 | $2817^{18}$ | 774,9 | 796,8 | 5 | 0 | $1523^{26}$ | 427,2 | 449,2 | 3 | ? |
| 8 | $500^{8}$ | 174,8 | 8 | 2 | 19,2 | 7 | $3500^{16}$ | 1236,6 | 1255,8 | 7 | 0 | $2400^{31}$ | 881,6 | 900,8 | 4 | ? |
| 9 | $500^{5}$ | 228,9 | 5 | 2 | 13,2 | 4 | $1509^{9}$ | 693,1 | 706,2 | 3 | 0 | $567^{8}$ | 265,4 | 278,6 | 1 | ? |
| 10 | $500^{7}$ | 181,3 | 7 | 4 | 19,9 | 7 | $3500^{18}$ | 1291,7 | 1311,6 | 7 | 1 | $1793^{28}$ | 671,7 | 691,6 | 3 | ? |
| 11 | $500^{10}$ | 237,1 | 13 | 4 | 47,7 | 12 | $6000^{25}$ | 2847,8 | 2895,5 | 12 | 0 | $2001^{44}$ | 983,7 | 1031,4 | 3 | ? |
| 12 | $500^{9}$ | 302,5 | 9 | 5 | 41,4 | 9 | $4003^{25}$ | 2589,4 | 2630,7 | 8 | 1 | $2114^{37}$ | 1331,8 | 1373,2 | 4 | ? |
| 13 | $500^{6}$ | 227,4 | 8 | 1 | 27,8 | 7 | $3007^{10}$ | 1355,8 | 1383,6 | 6 | 0 | $1104^{25}$ | 511,8 | 539,7 | 2 | ? |
| 14 | $500^{6}$ | 307,8 | 9 | 6 | 59,3 | 8 | $4000^{20}$ | 2468,5 | 2527,8 | 8 | 0 | $955^{28}$ | 610,0 | 669,3 | 0 | ? |
| 15 | $500^{5}$ | 380,9 | 9 | 1 | 42,0 | 8 | $3089^{18}$ | 2325,4 | 2367,5 | 6 | 0 | $703^{28}$ | 546,2 | 588,2 | 1 | ? |
| 16 | $500^{7}$ | 282,7 | 6 | 0 | 18,6 | 5 | $2002^{16}$ | 1127,3 | 1146,0 | 4 | 0 | $1093^{21}$ | 642,5 | 661,1 | 2 | ? |
| 17 | $500^{11}$ | 387,5 | 6 | 1 | 15,7 | 5 | $2004^{16}$ | 1545,2 | 1561,0 | 4 | 0 | $1037^{17}$ | 819,1 | 834,9 | 2 | ? |
| 18 | $500^{9}$ | 465,4 | 5 | 3 | 55,2 | 4 | $2000^{20}$ | 1886,5 | 1941,7 | 4 | 2 | $1507^{21}$ | 1415,8 | 1471,0 | 3 | ? |
| 19 | $500^{5}$ | 327,3 | 7 | 2 | 36,2 | 6 | $2502^{17}$ | 1634,3 | 1670,5 | 5 | 0 | $1058^{25}$ | 708,8 | 745,0 | 2 | ? |
| 20 | $500^{4}$ | 440,6 | 8 | 4 | 76,0 | 7 | $2687^{16}$ | 2361,5 | 2437,5 | 5 | 0 | $1099^{17}$ | 989,7 | 1065,6 | 2 | ? |
| 21 | $500^{7}$ | 548,9 | 7 | 2 | 72,7 | 6 | $3000^{19}$ | 3304,4 | 3377,0 | 6 | 0 | $1682^{30}$ | 1873,0 | 1945,7 | 2 | ? |
| 22 | $500^{8}$ | 372,9 | 7 | 3 | 49,3 | 6 | $2054^{18}$ | 1553,9 | 1603,2 | 4 | 0 | $533^{19}$ | 409,1 | 458,4 | 1 | ? |
| 23 | $500^{7}$ | 497,3 | 4 | 1 | 16,5 | 3 | $1500^{8}$ | 1495,3 | 1511,7 | 3 | 0 | $550^{17}$ | 560,7 | 577,2 | 1 | ? |
| 24 | $500^{9}$ | 651,8 | 15 | 6 | 193,1 | 14 | $7000^{38}$ | 9129,9 | 9323,0 | 14 | 0 | $2406^{50}$ | 3238,4 | 3431,5 | 4 | ? |
| 25 | $500^{6}$ | 471,0 | 4 | 0 | 15,6 | 3 | $1004^{7}$ | 945,3 | 960,8 | 2 | 0 | $508^{11}$ | 488,7 | 504,3 | 1 | ? |
| 26 | $500^{6}$ | 565,3 | 5 | 1 | 28,1 | 4 | $2000^{13}$ | 2255,5 | 2283,6 | 4 | 0 | $1028^{15}$ | 1194,5 | 1222,5 | 2 | ? |
| 27 | $500^{4}$ | 739,6 | 12 | 5 | 135,4 | 11 | $5500^{26}$ | 8032,2 | 8167,6 | 11 | 0 | $1657^{34}$ | 2446,9 | 2582,3 | 3 | ? |
| 28 | $500^{4}$ | 476,5 | 5 | 0 | 21,7 | 4 | $2000^{13}$ | 1905,8 | 1927,5 | 4 | 0 | $618^{18}$ | 605,6 | 627,2 | 1 | ? |
| 29 | $500^{8}$ | 657,6 | 7 | 5 | 67,6 | 6 | $2802^{24}$ | 3685,9 | 3753,5 | 5 | 0 | $958^{41}$ | 1288,8 | 1356,4 | 1 | ? |
| 30 | $500^{7}$ | 824,5 | 5 | 1 | 46,5 | 4 | $1008^{13}$ | 1627,8 | 1674,3 | 2 | 0 | $590^{17}$ | 974,0 | 1020,6 | 1 | ? |

Table 5: Results when testing on general dense hypergraphs with two distance functions. CPU times are reported in seconds.

The results in the dense case (hypergraphs 1-30) are given in Table 5. Column 'Nonextreme' contains the number of nonextreme points removed from the ordered nondominated set $\Phi$ when first phase stops. In column 'Unfinished' the number of triangles when the second phase stops before it reaches the upper bound are shown (a maximum of 500 *k'th* best solutions picked in each triangle are allowed). Column 'New extreme' contains the number of new supported extreme points found during the second phase. Note that these points do not necessarily have to be extreme points of the nondominated set when the procedure stops, as can be seen in Figure 25(b) where the frontier points of hypergraph two are plotted. The filled points are the points found by first phase, and the two other points are the two new extreme points found during the two-phases lower bound procedure. When the triangle

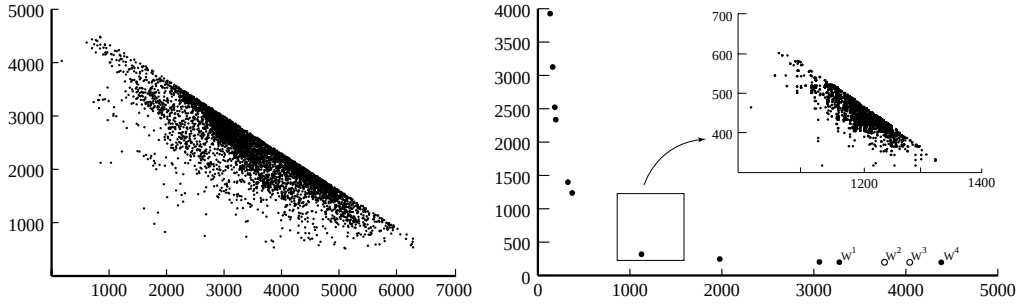
| | $k'th$ diag. (lb) | | 1. Phase | | | Phase Two (lb) | | | | | | Phase Two (ub) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hgraph | HPaths | CPU | Extreme | Nonfrontier | CPU | Triangles | HPaths | CPU | CPU total | Unfinished | New extreme | HPaths | CPU | CPU total | Unfinished | Nondom. |
| 31 | $500^{3}$ | 49,2 | 6 | 2 | 3,4 | 5 | $2500^{8}$ | 241,8 | 245,1 | 5 | 0 | $1580^{14}$ | 146,9 | 150,3 | 3 | ? |
| 32 | $500^{4}$ | 56,5 | 6 | 2 | 3,4 | 5 | $2500^{9}$ | 303,7 | 307,1 | 5 | 0 | $1553^{15}$ | 172,1 | 175,5 | 1 | ? |
| 33 | $500^{4}$ | 71,5 | 5 | 5 | 7,7 | 5 | $2002^{8}$ | 318,7 | 326,4 | 4 | 1 | $1007^{17}$ | 156,0 | 163,7 | 2 | ? |
| 34 | 1 | 0,4 | 1 | 0 | 0,4 | 0 | 0 | 0,0 | 0,4 | 0 | 0 | $0^{1}$ | 0,0 | 0,4 | 0 | 1 |
| 35 | $500^{3}$ | 117,0 | 3 | 0 | 3,0 | 2 | 294 | 66,1 | 69,0 | 0 | 0 | $20^{6}$ | 4,5 | 7,4 | 0 | 6 |
| 36 | $500^{5}$ | 171,7 | 7 | 4 | 16,4 | 6 | $3000^{10}$ | 1006,4 | 1022,8 | 6 | 0 | $2005^{15}$ | 658,5 | 674,9 | 4 | ? |
| 37 | $500^{4}$ | 175,7 | 4 | 0 | 3,5 | 3 | $1010^{4}$ | 355,8 | 359,3 | 2 | 0 | $393^{5}$ | 121,0 | 124,5 | 0 | ? |
| 38 | $500^{4}$ | 198,3 | 5 | 0 | 11,8 | 5 | $1028^{7}$ | 454,3 | 466,1 | 2 | 1 | $65^{8}$ | 23,7 | 35,5 | 0 | ? |
| 39 | $500^{4}$ | 264,0 | 2 | 0 | 1,8 | 1 | $500^{4}$ | 257,1 | 259,0 | 1 | 0 | $8^{4}$ | 3,8 | 5,6 | 0 | ? |
| 40 | 3 | 2,1 | 2 | 0 | 1,6 | 1 | 2 | 0,9 | 2,5 | 0 | 0 | $2^{2}$ | 0,9 | 2,5 | 0 | 2 |
| 41 | $500^{5}$ | 255,9 | 2 | 1 | 5,9 | 1 | $500^{5}$ | 255,1 | 261,0 | 1 | 0 | $68^{6}$ | 34,0 | 39,9 | 0 | ? |
| 42 | $500^{4}$ | 350,0 | 3 | 0 | 6,4 | 2 | $506^{4}$ | 364,7 | 371,1 | 1 | 0 | $8^{4}$ | 5,4 | 11,7 | 0 | ? |
| 43 | 4 | 3,5 | 2 | 0 | 2,3 | 1 | 3 | 1,8 | 4,1 | 0 | 0 | $3^{3}$ | 1,8 | 4,1 | 0 | 3 |
| 44 | 1 | 1,3 | 1 | 0 | 1,3 | 0 | 0 | 0,0 | 1,3 | 0 | 0 | $0^{1}$ | 0,0 | 1,3 | 0 | 1 |
| 45 | $500^{3}$ | 425,1 | 4 | 1 | 14,9 | 3 | $1004^{6}$ | 874,9 | 889,8 | 2 | 0 | $148^{9}$ | 127,6 | 142,5 | 0 | ? |
| 46 | 1 | 1,3 | 1 | 0 | 1,3 | 0 | 0 | 0,0 | 1,3 | 0 | 0 | $0^{1}$ | 0,0 | 1,3 | 0 | 1 |
| 47 | 1 | 1,6 | 1 | 0 | 1,6 | 0 | 0 | 0,0 | 1,6 | 0 | 0 | $0^{1}$ | 0,0 | 1,6 | 0 | 1 |
| 48 | $500^{4}$ | 615,9 | 4 | 0 | 21,0 | 3 | 9 | 9,8 | 30,8 | 0 | 0 | $9^{4}$ | 10,0 | 31,0 | 0 | 4 |
| 49 | 3 | 3,9 | 2 | 0 | 3,1 | 1 | 2 | 1,7 | 4,8 | 0 | 0 | $2^{2}$ | 1,7 | 4,8 | 0 | 2 |
| 50 | $500^{8}$ | 503,9 | 3 | 1 | 11,1 | 2 | 51 | 47,6 | 58,6 | 0 | 0 | $11^{8}$ | 10,6 | 21,6 | 0 | 8 |
| 51 | 1 | 2,4 | 1 | 0 | 2,4 | 0 | 0 | 0,0 | 2,4 | 0 | 0 | $0^{1}$ | 0,0 | 2,4 | 0 | 1 |
| 52 | 10 | 10,8 | 2 | 0 | 3,6 | 1 | 9 | 8,2 | 11,8 | 0 | 0 | $6^{4}$ | 5,6 | 9,2 | 0 | 4 |
| 53 | $500^{3}$ | 602,3 | 3 | 0 | 13,5 | 2 | 7 | 8,2 | 21,7 | 0 | 0 | $7^{3}$ | 8,3 | 21,8 | 0 | 3 |
| 54 | $500^{4}$ | 807,0 | 3 | 0 | 11,2 | 2 | $504^{4}$ | 808,7 | 820,0 | 1 | 0 | $21^{4}$ | 30,0 | 41,2 | 0 | ? |
| 55 | 3 | 5,2 | 2 | 0 | 4,1 | 1 | 2 | 2,2 | 6,3 | 0 | 0 | $2^{2}$ | 2,2 | 6,4 | 0 | 2 |
| 56 | 4 | 7,4 | 3 | 0 | 19,5 | 2 | 5 | 6,4 | 25,9 | 0 | 0 | $5^{3}$ | 6,5 | 26,0 | 0 | 3 |
| 57 | $500^{4}$ | 821,2 | 3 | 0 | 12,7 | 2 | $502^{4}$ | 825,8 | 838,5 | 1 | 0 | $39^{5}$ | 63,4 | 76,1 | 0 | ? |
| 58 | 14 | 20,1 | 4 | 0 | 18,6 | 3 | 7 | 8,8 | 27,4 | 0 | 0 | $7^{4}$ | 9,0 | 27,6 | 0 | 4 |
| 59 | 3 | 6,9 | 2 | 0 | 5,5 | 1 | 2 | 2,9 | 8,4 | 0 | 0 | $2^{2}$ | 3,0 | 8,5 | 0 | 2 |
| 60 | 7 | 16,4 | 4 | 0 | 29,1 | 3 | 7 | 13,5 | 42,5 | 0 | 0 | $7^{4}$ | 13,6 | 42,7 | 0 | 4 |

Table 6: Results when testing on general sparse hypergraphs with two distance functions. CPU times are reported in seconds.

defined by $W^1$ and $W^4$ is searched point $W^2$ is below the line between the two points a hence a new extreme point. However, when later the new lower/right point $W^3$ is found, $W^2$ is no longer a supported extreme because it is above the line between $W^1$ and $W^3$.

Finally, column 'Triangles' contains the total number of triangles searched during second phase. If the procedure stops before all nondominated points are found then the number of nondominated points are shown as a small number raised in the 'Hpaths' column.

As can be seen, it is much harder to solve the bi-SBT problem, when we consider two distance functions. None of the procedures found all nondominated points when hypergraphs $1-30$ were considered. This is due to the distribution of the points, as can be seen in Figure 25 when hypergraph 2 is considered. Here 5000 hyperpaths have been calculated in the direction




(a) Two sum functions

(b) Two distance functions

Figure 25: 5000 points plotted for hypergraph 2.

of the normal to the line between the upper/left and lower/right point. Figure 25(a) shows that if we consider two sum functions then all nondominated points are among the 5000 points. On the other hand, if we consider two distance functions and calculate 5000 points then they are all inside the square shown in Figure 25(b). Therefore two distance functions gives a much higher density of points inside the triangles.
Consider the two-phases procedure, here first phase is very fast compared to the second phase and the approximation found is good; only in 7 out of 30 hypergraphs, new extreme points were found. As can be seen from a comparison of procedure 'Two-phases (lb)' and 'Two-phases (ub)' the lower bound procedure does not perform well. This is due to the fact that the parametric lower bound calculated is very weak compared to the actual parametric weight. Actually, more than 80% of the hyperpaths found when we search a triangle have an actual parametric weight above the upper bound. On the contrary, if we use the upper bound function instead the CPU time is significant lower and considerably more nondominated points are found. The upper bound procedure on hypergraph one, for instance, found 42 nondominated points, and if the lower bound function is used, we only find 21 and have to use considerably more time. Because of the long CPU times, we could as a possible alternative to the methods described above use an interactive approach which assist the decision maker. For instance, first an approximation of the supported extreme points could be found with the first phase. The decision maker then could choose a triangle which is searched using the parametric weighting function, if all nondominated points must be found, or alternatively the upper bound weighting function resulting in a lower computation time.
Table 6 presents the same results for sparse hypergraphs (hypergraphs 31-60). Here all nondominated points were found in approximately half of the cases. The upper bound procedure still performs best and it finds more nondominated points. Moreover, when all nondominated points are found the upper bound procedure finds these points too.

## 6.3 Acyclic hypergraphs

We consider acyclic hypergraphs and hence can use the acyclic SBT procedure (Procedure 4). This procedure use a valid ordering which is obtained from B-MAKER as the numbering of the nodes. Since no heap is used, this procedure is faster than the general SBT procedure (Procedure 3). Moreover, branching on lower bounds now finds the true minimal weight which also will improve the CPU time.
We test the *k'th* diagonal (lb) and Two-phases (lb) procedure against the following acyclic procedures

***k'th* diagonal (acyclic)** The same as in *k'th* diagonal (lb), but use the *k'th* acyclic procedure instead (Procedure 9).

| | $k'th$ diagonal (lb) | | $k'th$ diagonal (acyclic) | | 1. phase | | | Two (acyclic) | | | | Two (lb) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hgraph | Hpaths | CPU | Hpaths | CPU | Extreme | CPU general | CPU acyclic | Triangles | Hpaths | CPU | CPU total | CPU total | Nondom. |
| 1 | 44 | 2,63 | 44 | 0,48 | 4 | 0,42 | 0,36 | 3 | 14 | 0,29 | 0,65 | 1,10 | 7 |
| 2 | $300^{12}$ | 25,44 | 1358 | 16,45 | 6 | 1,02 | 0,78 | 5 | 50 | 1,00 | 1,78 | 5,01 | 12 |
| 3 | 80 | 8,44 | 78 | 1,48 | 5 | 1,01 | 0,79 | 4 | 38 | 0,98 | 1,77 | 4,66 | 8 |
| 4 | 16 | 2,40 | 16 | 0,60 | 4 | 1,01 | 0,75 | 3 | 19 | 0,70 | 1,45 | 3,27 | 6 |
| 5 | 31 | 6,33 | 29 | 1,12 | 4 | 1,45 | 1,02 | 3 | 17 | 0,88 | 1,90 | 4,28 | 4 |
| 6 | 20 | 5,37 | 20 | 1,09 | 3 | 1,31 | 0,90 | 2 | 15 | 0,82 | 1,72 | 4,58 | 6 |
| 7 | 4 | 1,37 | 4 | 0,56 | 3 | 1,18 | 0,80 | 2 | 7 | 0,53 | 1,33 | 2,39 | 3 |
| 8 | 203 | 57,51 | 203 | 7,96 | 5 | 2,75 | 1,90 | 4 | 36 | 2,18 | 4,08 | 12,03 | 9 |
| 9 | 19 | 8,35 | 19 | 1,59 | 4 | 2,96 | 1,92 | 3 | 20 | 1,71 | 3,63 | 9,88 | 5 |
| 10 | 19 | 6,58 | 19 | 1,32 | 4 | 2,35 | 1,49 | 3 | 17 | 1,30 | 2,79 | 6,93 | 6 |
| 11 | $300^{12}$ | 140,38 | $10000^{12}$ | 488,67 | 6 | 5,50 | 3,23 | 5 | 35 | 3,26 | 6,49 | 19,97 | 12 |
| 12 | 11 | 6,93 | 11 | 1,66 | 4 | 3,97 | 2,53 | 3 | 15 | 1,98 | 4,51 | 10,61 | 8 |
| 13 | 56 | 20,58 | 56 | 3,31 | 4 | 2,71 | 1,82 | 3 | 35 | 2,42 | 4,24 | 14,34 | 7 |
| 14 | 12 | 7,31 | 12 | 1,72 | 3 | 2,76 | 1,82 | 2 | 15 | 1,67 | 3,49 | 9,88 | 6 |
| 15 | $300^{10}$ | 203,40 | 1046 | 78,45 | 6 | 8,04 | 5,00 | 5 | 27 | 4,27 | 9,27 | 23,62 | 10 |
| 16 | 25 | 13,95 | 25 | 2,32 | 4 | 3,84 | 2,29 | 3 | 29 | 2,63 | 4,92 | 17,50 | 10 |
| 17 | 17 | 12,77 | 17 | 2,46 | 5 | 6,42 | 3,94 | 4 | 31 | 4,02 | 7,96 | 24,84 | 9 |
| 18 | $300^{13}$ | 242,28 | 3729 | 326,50 | 6 | 9,55 | 5,97 | 5 | 70 | 8,91 | 14,88 | 62,67 | 14 |
| 19 | 18 | 10,85 | 18 | 2,21 | 5 | 5,21 | 3,35 | 4 | 24 | 3,07 | 6,42 | 16,44 | 7 |
| 20 | 13 | 11,49 | 13 | 2,52 | 4 | 5,69 | 3,55 | 3 | 15 | 2,80 | 6,35 | 15,22 | 8 |
| 21 | 25 | 26,02 | 25 | 4,31 | 4 | 7,19 | 4,41 | 3 | 25 | 4,45 | 8,86 | 28,80 | 7 |
| 22 | 15 | 10,85 | 15 | 2,31 | 3 | 3,39 | 2,13 | 2 | 29 | 3,01 | 5,14 | 20,72 | 8 |
| 23 | 41 | 37,57 | 41 | 5,59 | 6 | 10,31 | 6,37 | 5 | 30 | 5,83 | 12,20 | 33,00 | 9 |
| 24 | 16 | 21,81 | 16 | 3,91 | 6 | 14,14 | 7,99 | 5 | 35 | 7,76 | 15,75 | 51,21 | 10 |
| 25 | 26 | 20,73 | 26 | 3,53 | 4 | 5,52 | 3,42 | 3 | 23 | 3,40 | 6,82 | 20,72 | 6 |
| 26 | 127 | 134,82 | 127 | 15,67 | 4 | 7,82 | 4,83 | 3 | 29 | 5,33 | 10,16 | 35,87 | 7 |
| 27 | $300^{10}$ | 420,20 | 541 | 76,06 | 7 | 17,62 | 10,60 | 6 | 43 | 10,55 | 21,15 | 65,65 | 10 |
| 28 | 35 | 29,33 | 35 | 4,71 | 5 | 7,67 | 4,80 | 4 | 42 | 6,05 | 10,85 | 38,38 | 9 |
| 29 | $300^{12}$ | 340,02 | 705 | 86,53 | 7 | 15,85 | 9,59 | 6 | 48 | 10,25 | 19,84 | 64,42 | 13 |
| 30 | 175 | 246,64 | 175 | 27,92 | 5 | 13,52 | 8,15 | 4 | 21 | 6,69 | 14,84 | 38,37 | 8 |

Table 7: Results when testing on acyclic dense hypergraphs with two sum functions. CPU times are reported in seconds.

**Two-phases (acyclic)** We here again consider the two-phases procedures in Section 4.2. First, we use the phase one procedure (Procedure 12) in Section 4.2.1, together with an acyclic SBT procedure (Procedure 4). Next, we use the second phase procedure (Procedure 14) in Section 4.2.2, together with the *k'th* acyclic procedure (Procedure 9).

We do not consider the sparse case since the acyclic hypergraphs generated are so sparse that the procedures find all nondominated points in less than a second. This is the case for both two sum functions and two distance functions.
If we consider the dense case with two sum functions, the results of hypergraph 1-30 are shown in Table 7. As can be seen, the $k'th$ diagonal (acyclic) procedure performs best if it

| | 1. Phase | | | | Two (lb) | Phase Two (acyclic) | | | | | | Phase 2 (acyclic-ub) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hgraph | Extreme | Nonfrontier | CPU General | CPU acyclic | CPU total | HPaths | CPU | CPU total | Triangles | Unfinished | New extreme | HPaths | CPU | CPU total | Unfinished | Nondom. |
| 1 | 5 | 2 | 2,0 | 1,7 | 128,8 | $2000^{11}$ | 23,2 | 24,9 | 4 | 4 | 0 | $1036^{16}$ | 13,6 | 15,27 | 1 | ? |
| 2 | 4 | 2 | 1,9 | 1,6 | 104,3 | $1152^{15}$ | 16,5 | 18,1 | 3 | 2 | 0 | $558^{14}$ | 8,7 | 10,28 | 1 | ? |
| 3 | 2 | 1 | 1,0 | 0,8 | 59,2 | $500^{7}$ | 8,3 | 9,1 | 1 | 1 | 0 | $500^{12}$ | 8,8 | 9,56 | 1 | ? |
| 4 | 6 | 1 | 4,4 | 3,0 | 225,8 | $1503^{10}$ | 34,7 | 37,7 | 4 | 3 | 2 | $562^{10}$ | 12,7 | 15,63 | 1 | ? |
| 5 | 5 | 1 | 6,4 | 4,8 | 133,9 | $600^{9}$ | 19,4 | 24,1 | 4 | 1 | 0 | $514^{10}$ | 15,5 | 20,21 | 1 | ? |
| 6 | 9 | 2 | 15,6 | 11,5 | 1086,2 | $4000^{13}$ | 150,9 | 162,3 | 8 | 8 | 0 | $318^{26}$ | 12,9 | 24,31 | 0 | ? |
| 7 | 3 | 1 | 3,5 | 2,5 | 241,2 | $1000^{9}$ | 33,4 | 35,9 | 2 | 2 | 0 | $591^{8}$ | 20,4 | 22,93 | 1 | ? |
| 8 | 8 | 3 | 17,0 | 12,3 | 925,7 | $3024^{17}$ | 127,4 | 139,7 | 8 | 6 | 2 | $860^{19}$ | 37,3 | 49,52 | 1 | ? |
| 9 | 6 | 4 | 17,8 | 12,5 | 868,9 | $2100^{25}$ | 108,4 | 120,8 | 5 | 4 | 0 | $604^{29}$ | 30,9 | 43,40 | 1 | ? |
| 10 | 5 | 1 | 11,1 | 7,8 | 490,1 | $1530^{11}$ | 68,2 | 76,0 | 6 | 3 | 2 | $536^{12}$ | 23,0 | 30,85 | 1 | ? |
| 11 | 8 | 4 | 28,2 | 19,0 | 933,9 | $2016^{20}$ | 109,6 | 128,6 | 7 | 4 | 0 | $108^{22}$ | 8,0 | 27,03 | 0 | ? |
| 12 | 9 | 6 | 49,0 | 24,2 | 1797,0 | $3054^{22}$ | 208,3 | 232,6 | 8 | 6 | 3 | $995^{23}$ | 68,2 | 92,41 | 1 | ? |
| 13 | 7 | 5 | 22,2 | 15,5 | 1147,4 | $3000^{17}$ | 150,6 | 166,0 | 6 | 6 | 0 | $600^{23}$ | 31,4 | 46,86 | 1 | ? |
| 14 | 6 | 0 | 20,5 | 13,5 | 747,1 | $1194^{14}$ | 80,9 | 94,4 | 5 | 2 | 0 | $520^{17}$ | 34,9 | 48,45 | 1 | ? |
| 15 | 13 | 2 | 70,1 | 46,3 | 3836,7 | $5185^{21}$ | 441,8 | 488,1 | 12 | 10 | 0 | $1023^{23}$ | 89,1 | 135,42 | 1 | ? |
| 16 | 5 | 0 | 12,7 | 8,5 | 468,4 | $902^{8}$ | 55,3 | 63,8 | 4 | 1 | 0 | $513^{14}$ | 31,2 | 39,73 | 1 | ? |
| 17 | 6 | 3 | 30,2 | 19,9 | 775,8 | $1083^{12}$ | 88,3 | 108,2 | 5 | 2 | 0 | $535^{15}$ | 42,2 | 62,10 | 1 | ? |
| 18 | 8 | 1 | 40,0 | 25,3 | 2189,9 | $2399^{21}$ | 246,5 | 271,8 | 7 | 4 | 0 | $660^{26}$ | 65,5 | 90,86 | 1 | ? |
| 19 | 4 | 0 | 9,8 | 6,4 | 593,3 | $1003^{8}$ | 71,5 | 77,9 | 3 | 2 | 0 | $519^{7}$ | 38,1 | 44,55 | 1 | ? |
| 20 | 8 | 1 | 50,2 | 33,0 | 1600,2 | $1916^{19}$ | 179,1 | 212,1 | 7 | 3 | 0 | $551^{20}$ | 55,1 | 88,05 | 1 | ? |
| 21 | 4 | 1 | 17,3 | 11,4 | 1076,0 | $1017^{14}$ | 116,6 | 128,1 | 3 | 2 | 0 | $642^{23}$ | 72,7 | 84,14 | 1 | ? |
| 22 | 4 | 0 | 8,4 | 5,6 | 980,1 | $1462^{7}$ | 118,4 | 123,9 | 4 | 2 | 1 | $524^{7}$ | 42,3 | 47,90 | 1 | ? |
| 23 | 5 | 2 | 28,8 | 18,6 | 1044,4 | $1109^{11}$ | 121,3 | 139,9 | 4 | 2 | 0 | $514^{11}$ | 56,5 | 75,07 | 1 | ? |
| 24 | 5 | 1 | 35,0 | 22,6 | 1284,8 | $1045^{12}$ | 134,6 | 157,2 | 4 | 2 | 0 | $634^{18}$ | 81,9 | 104,49 | 1 | ? |
| 25 | 4 | 2 | 19,3 | 12,7 | 1171,3 | $1500^{10}$ | 135,6 | 148,3 | 3 | 3 | 0 | $983^{12}$ | 87,4 | 100,14 | 1 | ? |
| 26 | 6 | 0 | 45,7 | 28,4 | 1341,3 | $1214^{11}$ | 141,7 | 170,1 | 5 | 2 | 0 | $580^{13}$ | 68,7 | 97,11 | 1 | ? |
| 27 | 9 | 1 | 59,0 | 38,0 | 3527,0 | $2580^{17}$ | 376,5 | 414,5 | 8 | 5 | 0 | $645^{21}$ | 98,3 | 136,29 | 1 | ? |
| 28 | 5 | 1 | 24,5 | 16,0 | 897,4 | $1043^{12}$ | 104,5 | 120,5 | 4 | 2 | 0 | $524^{13}$ | 55,9 | 71,89 | 1 | ? |
| 29 | 6 | 2 | 56,6 | 34,9 | 1867,0 | $1537^{18}$ | 206,7 | 241,6 | 6 | 3 | 1 | $527^{20}$ | 70,1 | 105,07 | 1 | ? |
| 30 | 8 | 3 | 80,0 | 48,3 | 4292,7 | $2769^{22}$ | 441,7 | 490,0 | 7 | 5 | 0 | $751^{26}$ | 123,4 | 171,71 | 1 | ? |

Table 8: Results when testing on acyclic dense hypergraphs with two distance functions. CPU times are reported in seconds.

has to pick less than 100 hyperpaths, even though that the second phase of the two-phases procedure picks less hyperpaths. This is due to the fact that the initialization phases of the two-phases procedures are more time consuming than searching more paths. Moreover, we have to run the first phase too. However, as could be expected the two-phases procedure is stable and outperforms the diagonal procedure in most cases. Comparing the two-phases (lb) procedure with the two-phases (acyclic) procedure shows that the CPU time is reduced by more than 50%.

The results for two distance functions are presented in Table 8. Like general hypergraphs, the upper bound function performs best, and the first phase finds a good approximation of the extreme nondominated points. Furthermore, comparing the two-phases (lb) procedure

with the two-phases (acyclic) procedure show that the CPU time is reduced by more than 80%.
The reduction in CPU time, when the acyclic procedures are used, is mainly due to the acyclic SBT procedure since no heap is needed. Moreover, the fact that branching on lower bounds gives us the true minimal weight also contribute to a reduction in the CPU time.

### 6.4 Results summary

In this section, we presented test results for 60 hypergraphs when two sum or two distance functions are considered. We can summarize the above results as follows:

1. Branching on lower bounds works well. More than 95% of the lower bounds found where equal to the true minimum weight. As a result using a lower bound *k'th* procedure to calculate nondominated points improve the CPU time dramatically.

2. The two-phases procedure performs best, except for sparse hypergraphs when considering two sum functions. However, the two-phases procedure is more stable.

3. It is possible to find all nondominated solutions in relatively short time when two sum functions are considered.

4. If we consider two distance functions the problem is harder to solve. This is due to a much higher density of points inside the areas the two-phases procedure have to search.

5. Because the parametric weighting function gives a week lower bound in the distance case, using the upper bound function instead gives better results. However, the upper bound function only finds an approximation.

6. First phase finds supported extreme nondominated points fast compared to the second phase. The first phase only finds an approximation in the distance case. However, this approximation is very good.

7. Using the acyclic procedures on acyclic hypergraphs give a high reduction in the CPU time. This is mainly due to the acyclic SBT procedure. Moreover, the fact that branching on lower bounds give us the true minimal weight also contribute to the reduction.

## 7 Applications

### 7.1 Random time dependent shortest paths

We consider *random time dependent networks* [24], i.e. the travel time through an arc is random and depend on the departure time. We assume that departure times are integer and travel times are integer-valued discrete random variables. Departure and arrival times belong to a finite *time horizon*, that is an interval $H = [0, ..., t_{\max}]$ of integer values. In practice we assume that the relevant time period is *discretized* into time intervals of length $\delta$, i.e. the time horizon $H$ corresponds to the set of time instants $\{0, \delta, ..., t_{\max}\delta\}$.
Let $G = (N, A)$ be the underlying digraph called *the topological network* and let $d \in N$ be the *destination* node. Assume that there are no arcs leaving node $d$. Let the following be defined for $G$

| | | | |
|---|---|---|---|
| $L(i,j)$ | Possible departure times from node $i$ along arc $(i,j)$ $(L(i,j) \subset H)$. | $L(i) = \bigcup_{(i,j)\in A} L(i,j),\ i \neq d$ | Possible departure times from node $i$ along some arc $(L(d) = H)$. |
| $K(i,j,t)$ | The number of possible arrival times at node $j$ when leaving node $i$ at time $t$. | $I(i,j,t) = \{t_1, ..., t_{K(i,j,t)}\}$ | The $K(i,j,t)$ possible arrival times at $j$ when leaving node $i$ at time $t$. |
| $p_{ijt}(t_h)$ | The probability of arriving at node $j$ at time $t_h$ when leaving node $i$ at time $t$. | $K = \sum_{(i,j)\in A, t\in L(i,j)} K(i,j,t)$ | Total number of possible travel times (inputsize). |

Moreover, assume that $G$ satisfies the following assumptions:

1. For each $(i,j) \in A$ and $t \in L(i,j)$ we have $I(i,j,t) \subseteq L(j)$.

2. Travel times are positive, i.e. $t_h > t$ for each $t_h \in I(i,j,t)$.

Assumption 1 ensures that the traveller cannot get stuck at a node, and since travel times are positive (assumption 2) and $t_{\max}$ is finite, a traveller arrives at node $d$ within time $t_{\max}$, travelling through at most $t_{\max}$ arcs.
A strategy is a function

$$S : (N \setminus \{d\} \times H) \to A$$

assigning to each pair $(i,t)$ with $t \in L(i)$ a *successor* arc $(i,j)$ with $t \in L(i,j)$. Formally speaking given a strategy $S$, the strategy tells that if we are leaving node $i$ at time $t$, we leave along arc $S(i,t)$.
Given a strategy $S$, the corresponding *expected arrival time* $E_T^S$ is defined by the following recursive equations:

$$E_T^S(i,t) = \begin{cases} \sum_{t_h \in I(i,j,t)} p_{ijt}(t_h) E_T^S(j,t_h) & S(i,t) = (i,j) \qquad i \neq d \\ t & i = d \end{cases}$$

*The minimum expected travel time problem* (MET) consists now in finding a strategy $S$ yielding a minimum $E_T^S(i,t)$ for each pair $(i,t)$ with $t \in L(i)$. Note that minimizing expected arrival time is the same as minimizing travel time because the travel time is given by $E_T^S(i,t) - t$.

**Example 8** Consider the topological network $G$ in Figure 26. The time horizon $H$ is

$$H = [0, ..., 6]$$

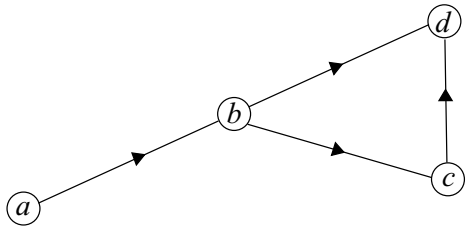



Figure 26: The topological network $G$.

and the departure times along the arcs are given by:

$$\begin{array}{ll} L(a,b) = \{0\} & L(b,c) = \{1,2\} \\ L(b,d) = \{1,2\} & L(c,d) = \{2,3\} \end{array}$$

The sets $I$ and the corresponding probabilities are given by:

| $(i,j),t$ | $(a,b),0$ | $(b,c),1$ | $(b,c),2$ | $(b,d),1$ | $(b,d),2$ | $(c,d),2$ | $(c,d),3$ |
|---|---|---|---|---|---|---|---|
| $I(i,j,t)$ | $\{1,2\}$ | $\{2\}$ | $\{3\}$ | $\{3\}$ | $\{5\}$ | $\{3,4\}$ | $\{4,6\}$ |
| $p_{ijt}$ | $\left\{\frac{1}{3},\frac{2}{3}\right\}$ | $\{1\}$ | $\{1\}$ | $\{1\}$ | $\{1\}$ | $\left\{\frac{1}{4},\frac{3}{4}\right\}$ | $\left\{\frac{3}{4},\frac{1}{4}\right\}$ |

If we e.g. leave node $c$ at time 2 along arc $(c,d)$ we will arrive at node $d$ at time 3 with a probability of $\frac{1}{4}$ or at time 4 with a probability of $\frac{3}{4}$.
A possible strategy $S_1$ is

| $(v,t)$ | $(a,0)$ | $(b,1)$ | $(b,2)$ | $(c,2)$ | $(c,3)$ |
|---|---|---|---|---|---|
| arc | $(a,b)$ | $(b,d)$ | $(b,c)$ | $(c,d)$ | $(c,d)$ |

If we leave node $a$ at time 0, the expected arrival time at node $d$ is

$$\begin{aligned} E_T^S(a,0) &= \sum_{t_h \in I(a,b,0)} p_{abt}(t_h) E_T^S(b,t_h) = \frac{1}{3}E_T^S(b,1) + \frac{2}{3}E_T^S(b,2) = \frac{1}{3}E_T^S(d,3) + \frac{2}{3}E_T^S(c,3) \\ &= \frac{1}{3}\cdot 3 + \frac{2}{3}\left(\frac{3}{4}\cdot 4 + \frac{1}{4}\cdot 6\right) = 4 \end{aligned}$$

■

### 7.1.1 The hypergraph model

In order to formulate MET in terms of shortest hyperpaths, we define the *time expanded hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ as follows

$$\begin{aligned} \mathcal{V} &= \{(i,t) \mid i \in N, t \in L(i)\} \cup \{s\} \\ h(i,j,t) &= \{\{(j,t_h) : t_h \in I(i,j,t)\}, (i,t)\} \quad \forall (i,j) \in A, t \in I(i,j) \\ h(d,t) &= (\{s\}, (d,t)) \qquad \forall t \in H \\ \mathcal{E} &= \{h(i,j,t) \mid (i,j) \in A, t \in L(i,j)\} \cup \{h(d,t) \mid t \in H\} \end{aligned}$$

$\mathcal{H}$ satisfies the following properties:

1. $\mathcal{V}$ contains one node for each pair $(i,t)$, $t \in L(i)$, plus an origin node $s$.
2. The orientation of $h(i,j,t)$ is opposite to the orientation of arc $(i,j)$, indeed the tail of $h(i,j,t)$ contains a pair $(j,t_h)$ for each possible arrival time at $j$ when leaving $i$ at time $t$, while the head contains the pair $(i,t)$. In addition for each $t \in H$, we have defined an arc $h(d,t)$ from the origin $s$ to node $(d,t)$.
3. $Size(\mathcal{H}) = \mathcal{O}(K)$, and $\mathcal{H}$ can be built in $\mathcal{O}(K)$ time.
4. $\mathcal{H}$ is an acyclic hypergraph with origin $s$ as a consequence of the assumptions of $G$.
5. There is a one to one correspondence between a strategy and a predecessor function $f$ on $\mathcal{H}$, that is

$$\begin{array}{c} f(i,t) = h(i,j,t) \\ \Updownarrow \\ S(i,t) = (i,j) \end{array}$$

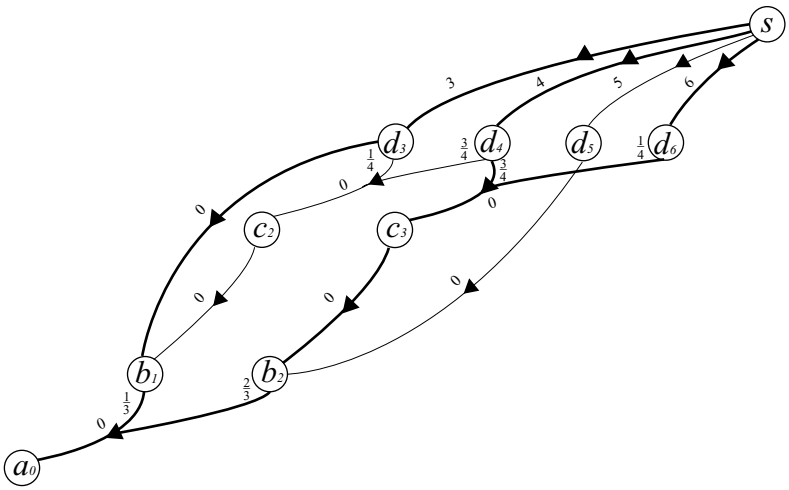



Figure 27: The time expanded hypergraph $\mathcal{H}$.

Given a strategy $S$, denote by $f^S$ the corresponding predecessor function on $\mathcal{H}$. Moreover, denote by $\pi^S_{(i,t)}$ the hyperpath from $s$ to $(i,t)$ defined by $f^S$. Note that hyperpath $\pi^S_{(i,t)}$ represents the route of a traveller leaving from $i$ at time $t$ and following strategy $S$. Let the weights of $\mathcal{H}$ be given by

$$w(e) = \begin{cases} t & \text{if } e = h(d,t) \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

and for each hyperarc $e \in \mathcal{E}$ assign multipliers

$$a_e(v) = \begin{cases} p_{ijt}(t_h) & \forall e = h(i,j,t) \quad \forall v = (j,t_h) \in T(e) \\ 1 & \forall e = h(d,t) \quad v = \{s\} \in T(e) \end{cases} \tag{7.2}$$

It is now possible to solve MET as a shortest hyperpath problem.

**Theorem 8 (Pretolani [24])** For each $i \in N$ and $t \in L(i)$ the expected arrival time $E^S_T(i,t)$ is equal to the value of hyperpath $\pi^S_{(i,t)}$, with respect to weights (7.1) and multipliers (7.2). Moreover, MET can be formulated as a minimum value problem on $\mathcal{H}$ and be solved in $\mathcal{O}(K)$ time.

**Example 8** (continued) The time expanded hypergraph is illustrated in Figure 27. Here $\mathcal{H}$ has 4 different hyperpaths from $s$ to $a_0$.
Strategy $S_1$ defines the hyperpath $\pi^{S_1}_{(a,0)}$ with weight $W\left(\pi^{S_1}_{(a,0)}\right) = 4$ emphasized in Figure 27. ■

The hypergraph model presented shows a high degree of flexibility. Other route selection criteria can be modeled by defining suitable weights and multipliers for the hypergraph model.

**Max possible arrival time** Given a strategy $S$, *the maximum possible arrival time* $M_T^S$ is defined by the recursive equations:

$$M_T^S(i,t) = \begin{cases} \max\limits_{t_h \in I(i,j,t)} M_T^S(j,t_h) & S(i,t) = (i,j),\ i \neq d \\ t & i = d \end{cases}$$

The *min-max travel time problem* (MMT) now consists in finding a strategy minimizing $M_T^S(i,t)$ for each $i$ and $t \in L(i)$. By comparing the definitions of $M_T^S$ and distance we have

**Theorem 9 (Pretolani [24])** For each $i \in N$ and $t \in L(i)$ the maximum possible arrival time $M_T^S(i,t)$ is equal to the distance of hyperpath $\pi_{(i,t)}^S$, with respect to the weights (7.1). Moreover, MMT can be formulated as a minimum distance problem on $\mathcal{H}$ and be solved in $\mathcal{O}(K)$ time.

**Expected cost** Given a strategy $S$, *the expected costs* $E_C^S$ is defined by the recursive equations:

$$E_C^S(i,t) = \begin{cases} c(i,j,t) + \sum\limits_{t_h \in I(i,j,t)} p_{ijt}(t_h) E_C^S(j,t_h) & S(i,t) = (i,j),\ i \neq d \\ g_d(t) & i = d \end{cases}$$

where $g_d(t)$ is a penalty function for early or late arrivals. *The minimum expected cost problem* (MEC) consists in finding a strategy $S$ yielding a minimum $E_C^S(i,t)$ for each pair $(i,t)$ with $t \in L(i)$. If we define the following weights

$$w(e) = \begin{cases} g_d(t) & e = h(d,t) \\ c(i,j,t) & e = h(i,j,t) \end{cases} \tag{7.3}$$

we have

**Theorem 10 (Pretolani [24])** For each $i \in N$ and $t \in L(i)$ the expected cost $E_C^S$ is equal to the value of hyperpath $\pi_{(i,t)}^S$, with respect to the weights (7.3) and multipliers (7.2). Moreover, MEC can be formulated as a minimum value problem on $\mathcal{H}$ and be solved in $\mathcal{O}(K)$ time.

**Max possible cost** Given a strategy $S$ *the maximum possible cost* $M_C^S$ is defined by the recursive equations:

$$M_C^S(i,t) = \begin{cases} \max\limits_{t_h \in I(i,j,t)} M_C^S(j,t_h) & S(i,t) = (i,j),\ i \neq d \\ g_d(t) & i = d \end{cases}$$

The *min-max travel time problem* (MMT) consists in finding a strategy minimizing $M_C^S(i,t)$ for each $i$ and $t \in L(i)$. Comparing the definition of $M_C^S$ and distance we have

**Theorem 11 (Pretolani [24])** For each $i \in N$ and $t \in L(i)$ the maximum possible cost $M_C^S(i,t)$ is equal to the distance of hyperpath $\pi_{(i,t)}^S$, with respect to the weights (7.3). Moreover, MMC can be formulated as a minimum distance problem on $\mathcal{H}$ and be solved in $\mathcal{O}(K)$ time.

**Example 8** (continued) Let the costs of hyperarc $h(i,j,t)$ be given by:

| $(i,j),t$ | $(a,b),0$ | $(b,c),1$ | $(b,c),2$ | $(b,d),1$ | $(b,d),2$ | $(c,d),2$ | $(c,d),3$ |
|---|---|---|---|---|---|---|---|
| $c(i,j,t)$ | 1 | 1 | 2 | 2 | 3 | 1 | 0 |

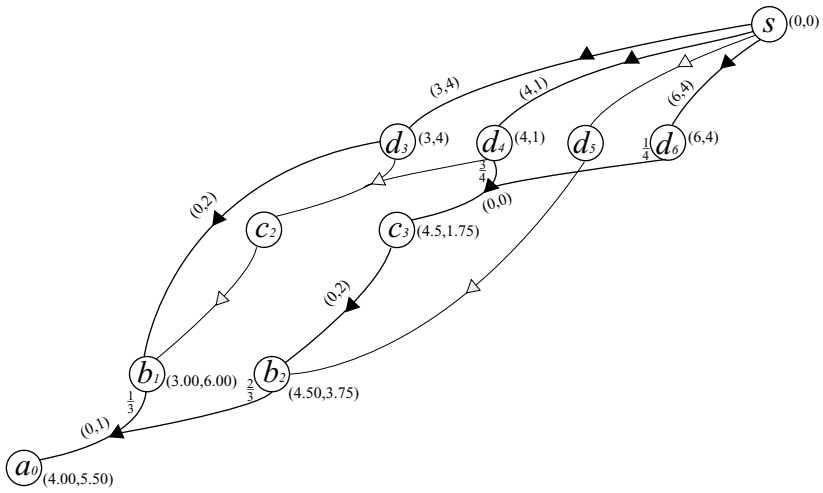



Figure 28: The first efficient hyperpath from $s$ to $a_0$ for MET and MEC.

and assume that the penalty cost is

| $t$ | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| $g_d(t)$ | 4 | 1 | 2 | 4 |

For $\pi^{S_1}_{(a,0)}$ we now have

$$E^{S_1}_C(a,0) = 5.5 \quad M^{S_1}_T(a,0) = 6 \quad M^{S_1}_C(a,0) = 7$$

■

### 7.1.2 Finding an efficient strategy

Suppose now that we want to find a strategy that solves both MET and MEC or solves both MMT and MMC. Normally a solution that minimizes both objectives does not exist, instead we have to find an efficient strategy. From the connection between strategy and hyperpath we have

**Theorem 12** Given a random time dependent network, an efficient strategy with respect to two criteria (e.g. expected cost and travel time) can be found by solving the bicriterion shortest hyperpath problem on the time expanded hypergraph $\mathcal{H}$.

**Example 8** (continued) Suppose we want to minimize both expected arrival time and cost, i.e. we have to solve the bi-SBT problem when two value functions are considered with weights and multipliers as in Figure 28. Since $\mathcal{H}$ is acyclic, the acyclic procedures in Section 4 can be used to find the two efficient hyperpaths shown in Figure 28 and Figure 29. Here the first and second weight w.r.t. the value function are shown beside each node in the curved parenthesis. If we want to minimize both maximum arrival time and cost we have to solve the bi-SBT problem with two distance functions. This gives only one efficient hyperpath which is shown in Figure 30. Here the first and second weight w.r.t. the distance function are shown beside each node in the square parenthesis. ■

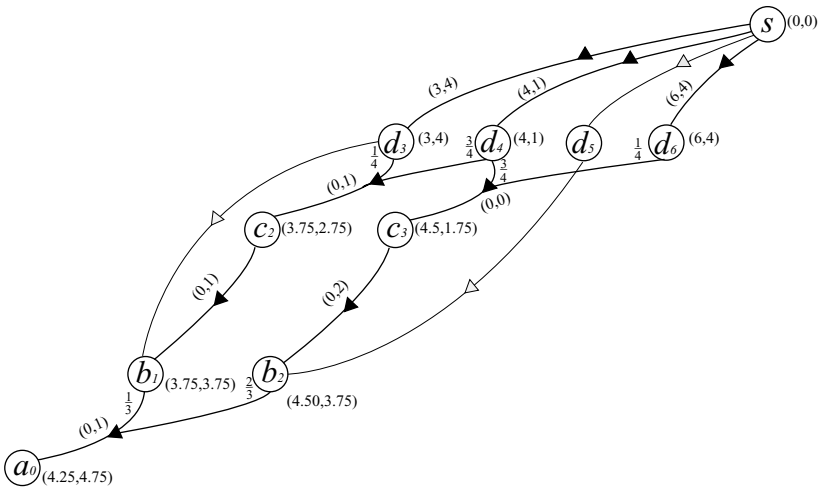



Figure 29: Second efficient hyperpath from $s$ to $a_0$ for MET and MEC.

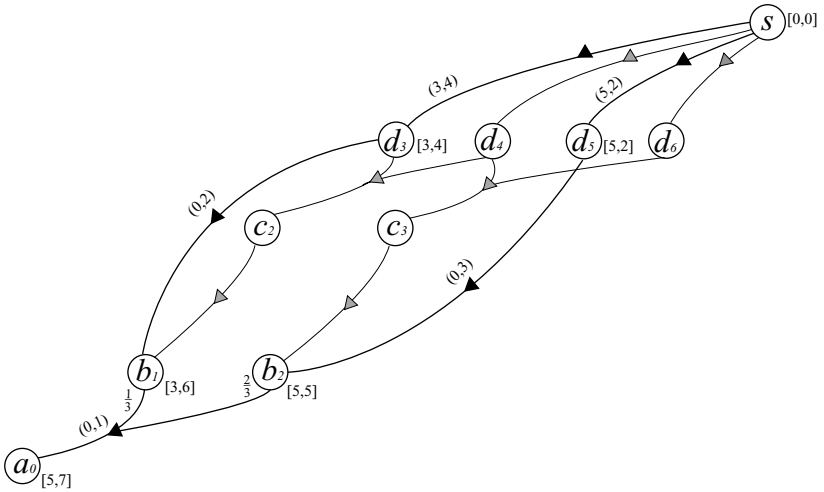



Figure 30: Efficient hyperpath from $s$ to $a_0$ for MMT and MMC.

# 8 Conclusions

In this paper we have defined the bi-SBT problem and presented different methods for solving the problem. We considered two cases of weighting functions: Two value weighting functions and two distance weighting functions.
All the methods use a *k'th* best procedure to search for efficient hyperpaths. This problem was extended from digraphs to hypergraphs. Moreover, we presented a method to find all hyperpaths with weight below an upper bound, by branching on lower bounds.
Computational testing revealed that, by branching on lower bounds and using a two-phases

approach, we could solve the bi-SBT problem fast, if two sum weighting functions were considered. If two distance functions are considered the problem is harder to solve. However, by using an upper bound weighting function a good approximation of the nondominated points could be found. Furthermore, it is possible to improve the CPU time dramatically, if acyclic hypergraphs are considered.
We have not developed procedures for solving the problem when a value and a distance weighting function are considered. Here it seems harder to find a lower bound of the minimal parametric weight of a subhypergraph. An upper bound approximation is properly needed. Moreover, it may be possible to find better ways to prune the search tree, or to improve the lower bound weighting function when two distance weighting functions are considered. These areas could be interesting directions for further research.

# Appendix A: Data structures

In this section we describe how the procedures in Section 2-4 were implemented in C++. It is assumed that the reader is familiar with pointers and structures/classes. Examples of program code is written in C++.

## A.1 Heap representation

We use a heap to sort the candidate sets, that is, the set is sorted using a binary tree. The heap is implemented using an array. The heap implementation is well-known and not stated here. The interested reader should refer to Tarjan [28].

## A.2 Hypergraph representation

Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = (v_1, ..., v_n)$ is the set of nodes, and $\mathcal{E} = (e_1, ..., e_m)$ is the set of hyperarcs. The hyperarcs $\mathcal{E}$ are divided into arcs $e$ and true hyperarcs $eh$ that is $\mathcal{E} = (e_1, ..., e_{ma}) \cup (eh_1, ..., eh_{mh})$. Let *htailsize* denote the total size of the tails of true hyperarcs. A hypergraph class/structure representing a forward and/or a backward representation now consists of arrays containing arc, hyperarc and node structures. What the arc, hyperarc and node structures contain depends on the use. For instance a *Node* class for calculating a shortest hyperpath could contain

| *Node* | |
|---|---|
| *weight(s)* | *pred* |
| *pAFirst* | *pHFirst* |

Here *weight(s)* is the optimal weight(s) for the shortest hyperpath and *pred* denote the predecessor hyperarc. Furthermore, *pAFirst* and *pHFirst* denote pointers to the first arc and hyperarc in the backward star of the node, respectively. The prefix *p* before a type denote that it is a pointer and *pp* denotes a pointer to a pointer. We now consider the backward and forward representation of a hypergraph.

### A.2.1 Backward Representation

Given a node $v$, the backward representation needs to know the hyperarcs belonging to $BS(v)$. Here an arc, hyperarc and node is defined by

| *Arc* |
|---|
| *length(s)* |
| *pHead* |
| *pTail* |

| *HArc* |
|---|
| *length(s)* |
| *pHead* |
| *ppTail* |

| *Node* |
|---|
| *weight(s)* |
| *pAFirst* |
| *pHFirst* |

The backward representation of a hypergraph can now be obtained by storing the arcs, hyperarcs and nodes in arrays which fulfill: The node array is an array of size $n+2$, where each entry contains a *Node.*

| 0 | 1 | | | | $n$ | $n+1$ |
|---|---|---|---|---|---|---|
| $D$ | $v_1$ | . | . | . | $v_n$ | $D$ |

Here node $v_i$ is stored in entry $i$. The zero entry is not used and entry $n+1$ is used as a dummy node ($D$ stands for *Dummy)* so it is possible to use *for* statements.
The arc array is an array of size $ma+2$, where each entry contains an *Arc.* The array is sorted in a backward star order, i.e. like (3.1). Again a dummy node is used.

| 0 | 1 | | | | $ma$ | $ma+1$ |
|---|---|---|---|---|---|---|
| $D$ | $e_1$ | . | . | . | $e_{ma}$ | $D$ |

If *pNode* is a pointer to a node then the $BS$ of arcs now can be scanned by using the following *for* statement

```
for(pArcNow = pNode->pAFirst, pLastArc = (pNode+1)->pAFirst;
    pArcNow!=pLastArc;
    pArcNow++)
{
    statements;
}
```

Similarly the hyperarc array have $mh+1$ *HArc* entries sorted in a backward star order.

| 0 | 1 | | | | $mh$ | $mh+1$ |
|---|---|---|---|---|---|---|
| $D$ | $eh_1$ | . | . | . | $eh_{mh}$ | $D$ |

Again if *pNode* is a pointer to a node, then the $BS$ of hyperarcs can be scanned by using the following *for* statement

```
for(pHArcNow = pNode->pHFirst, pLastHArc = (pNode+1)->pHFirst;
    pHArcNow!=pLastHArc;
    pHArcNow++)
{
    statements;
}
```

To store the tails of the hyperarcs we use a tail array with $htailsize+1$ entries, each containing a pointer to a *Node.*

| 0 | 1 | | | | $htailsize$ | $htailsize+1$ |
|---|---|---|---|---|---|---|
| $D$ | $p_1$ | . | . | . | $p_{htailsize}$ | $D$ |

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

```
for(ppNodeNow = pHArc->ppTail, ppLastNode = (pHArc+1)->ppTail;
    ppNodeNow!=ppLastNode;
    pNodeNow++)
{
    pNode = *ppNodeNow;
    statements;
}
```

### A.2.2 Backward and Forward Representation

If a forward representation is needed together with a backward representation two more arrays are needed. A forward star arc and forward star hyperarc array. The node must also be modified to contain two more pointers.

| *Node* |
|---|
| *weight(s) pAFirst* |
| *ppAFirst pHFirst* |
| *ppHFirst* |

The forward star arc array is of size $ma + 2$ and contain *Arc* pointers.

| 0 | 1 | | | | $ma$ | $ma+1$ |
|---|---|---|---|---|---|---|
| $D$ | $p_1$ | . | . | . | $p_{ma}$ | $D$ |

Here the forward star of arcs to a node pointed by *pNode* can be scanned by

```
for(ppArcNow = pNode->ppAFirst, ppLastArc = (pNode+1)->ppAFirst;
    ppArcNow!=ppLastArc;
    ppArcNow++)
{
    pArc = *ppArcNow;
    statements;
}
```

The forward star hyperarc array is of size $htailsize + 2$ and contains *HArc* pointers.

| 0 | 1 | | | | $tailsize$ | $tailsize+1$ |
|---|---|---|---|---|---|---|
| $D$ | $p_1$ | . | . | . | $p_{mh}$ | $D$ |

and if *pNode* is a pointer to a node, then the forward star of hyperarcs can be scanned by using the following *for* statement

```
for(ppHArcNow = pNode->ppHFirst, ppLastHArc = (pNode+1)->ppHFirst;
    ppHArcNow!=ppLastHArc;
    ppHArcNow++)
{
    pHArc = *ppHArcNow;
    statements;
}
```

### A.2.3 Forward Representation

If only a forward representation is needed, the forward star arc array can be deleted and the arc array sorted in a forward star order instead. Moreover, the hyperarc array does not have to be sorted in a backward star order.

## A.3 Branchingtree representation

The branching tree defining the subhypergraphs used in the *k'th* procedures were implemented with a dynamic branching tree. This is illustrated in Figure 31 where the branching tree, when we pick the $4'th$ shortest hyperpath in Example 3 on page 14, is shown. Here each branch node ($BNode_i$) corresponds to a subhypergraph $\mathcal{H}_i$ ($BNode_0$ corresponds to hypergraph $\mathcal{H}$). We assume that we use Branching Rule 2 on page 18. From $\mathcal{H}$ we generated 3 subhypergraphs $\mathcal{H}_1, \mathcal{H}_2$ and $\mathcal{H}_3$ and from $\mathcal{H}_1$ two more subhypergraphs $\mathcal{H}_{11}$ and $\mathcal{H}_{12}$ etc. In each subhypergraph we removed the last branching hyperarc and fixed the backward star for the other branching hyperarcs. Now if we for each branch node $BNode_i$ in the branching tree store the hyperarc we remove, then by traversing the path from $BNode_i$ to $BNode_0$, we

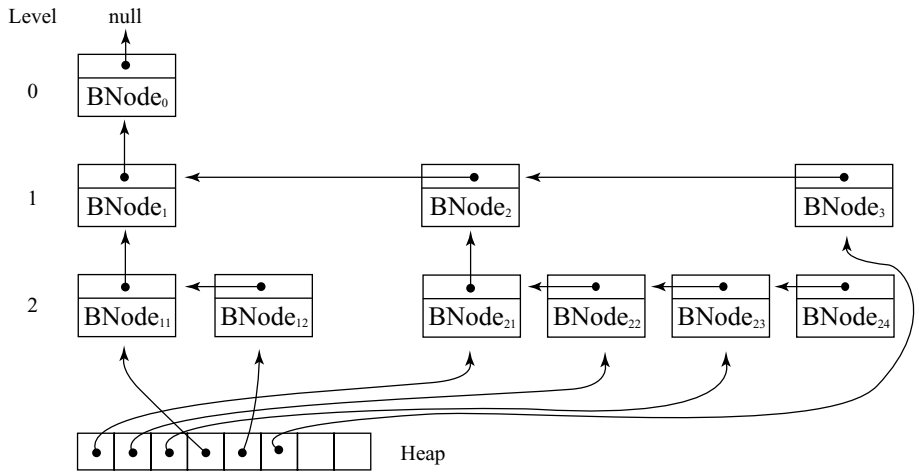



Figure 31: The branching tree corresponding to all subhypergraphs generated.

can construct subhypergraph $\mathcal{H}_i$. For instance subhypergraph $\mathcal{H}_{22}$ corresponds to the path $BNode_{22}$ - $BNode_0$ in Figure 31 because if the following is stored in the branch nodes

| $BNode_1$ | $BNode_2$ | $BNode_{21}$ | $BNode_{22}$ |
|---|---|---|---|
| $e_t^1$ | $e_6^1$ | $e_6^2$ | $e_3^2$ |

then we obtain $\mathcal{H}_{22}$ from $\mathcal{H}$ by removing $e_3^2$, set $BS\left(h\left(e_6^2\right)\right) = e_6^2$, remove $e_6^1$ and set $BS\left(h\left(e_t^1\right)\right) = e_t^1$. More generally consider path

$$BNode_i - BNode_{i-1} - ... - BNode_0$$

and let $e_i$ denote the hyperarc stored in $BNode_i$. Let $firstChild_i$ be true if node $BNode_i$ is on level $j$ and node $BNode_{i-1}$ is on level $j-1$ and otherwise false (see Figure 31). Then the path corresponds to the subhypergraph of $\mathcal{H}_i$ obtained from $\mathcal{H}$ by

1. Removing $e_i$
2. Setting $BS\left(h\left(e_q\right)\right) = e_q$ if $firstChild_{q+1} = false \quad (q \neq i)$
3. Removing $e_q$ if $firstChild_{q+1} = true \quad (q \neq i)$

Each $BNode_i$ is a structure containing

| *BNode* | |
|---|---|
| *weight* | *harc* |
| *firstChild* | *branchlist* |
| *pPrev* | |

where *weight* is the optimal weight for the shortest hyperpath in $\mathcal{H}_i$ (or a lower bound) and *harc* is the hyperarc we remove. The list *branchlist* contains the branching hyperarcs of the shortest hyperpath of $\mathcal{H}_i$, note if we branch on lower bounds, this list is not needed. Last *pPrev* is a pointer to the previous *BNode*.
The subhypergraph containing the minimal weight is found by storing all candidates in a heap of pointers (see Figure 31).

# References


[1] Josè A. Azevedo, Joaquim J. E. R. S. Madeira, and Ernesto Q. V. Martins. A computational improvement for a shortest paths ranking algorithm. *EJOR*, 73:188–191, 1994.

[2] Josè Augusto Azevedo, Maria E. O. S. Costa, Joaquim J. E. R. S. Maderira, and Ernesto Q. V. Martins. An algorithm for the ranking of shortest paths. *EJOR*, 69:97–106, 1993.

[3] J. Brambaugh-Smith and D. Shier. An empirical investigation af some bicriterion shortest path algorithms. *EJOR*, 43:216–224, 1989.

[4] R. Cambini, G Gallo, and M. G. Scutella. Flows om hypergraphs. *Mathematical Programming*, 78:195–217, 1997.

[5] J. C. N. Climaco and E. Q. V. Martins. A bicriterion shortest path algorithm. *EJOR*, 11:399–404, 1982.

[6] J. Cohen. *Multiobjective Programming and Planning*. Academic Press, New York, 1978.

[7] 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 and Operations Research*, 26:789–798, 1999.

[8] John R. Current, Charles S. ReVelle, and Jared L. Cohen. An interactive approach to identify the best compromise solution for two objective shortest path problems. *Computers and operational research*, 17(2):187–198, 1990.

[9] G. Gallo, G. Longo, S. Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied Mathematics*, 42:177–201, 1993.

[10] G. Gallo and S. Pallottino. Hypergraph models and algorithms for the assembly problem. Technical Report 6, Dipartimento di Informatica, March 1992.

[11] M. Garey and D. Johnson. *Computers and Intractability. A Guide of the Theory of NP-Completeness*. W. H. Freeman, 1979.

[12] R. W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[13] R. G. Jeroslow, K. Martin, R. L. Rardin, and J. Wang. Gainfree Leontief substitution flow problems. *Mathematical Programming*, 57:375–414, 1992.

[14] 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.

[15] R. De Leone and D. Pretolani. Auction algorithms for shortest hyperpath problems. Technical Report OR-CAM-1998-01, Dipartimento di Matematica e Fisica, aug 1998.

[16] Elise D. Miller-Hooks and Hani S. Mahmassani. Least possible time paths in stochastic, time-varying networks. Technical report, University of Texas at Austin, 1998.

[17] Elise D. Miller-Hooks and Hani S. Mahmassani. On the generation af nondominated paths in stochastic, time-varying networks. Technical report, University of Texas at Austin, 1998.

[18] Elise D. Miller-Hooks and Hani S. Mahmassani. Optimal routing of hazardous materials in stochastic, time-varying transportation networks. Technical report, University of Texas at Austin, 1998.

[19] J. Mote, I. Murthy, and D. L. Olson. A parametric approach to solving bicriterion shortest path problems. *EJOR*, 53:81–92, 1991.

[20] S. Nguyen and D. Pretolani. Shortest hyperpath problems on oriented hypergraphs. Technical report, Centre de recherche sur les transports, Sep 1994.

[21] S. Nguyen, D. Pretolani, and L. Markenzon. On some path problems on oriented hypergraphs. *RAIRO - Informatique theorique et appliqations*, (32 No. 1), 1998.

[22] Lars Relund Nielsen, Daniele Pretolani, and Kim Allan Andersen. A remark on the definition of a B-hyperpath. Technical report, Department of Operations Research, University of Aarhus, 2001. Available at http://www.imf.au.dk/∼relund/.

[23] G. H. Polychronopoulos and J. N. Tsitsiklis. Stochastic shortest path problems with recourse. *Networks*, 27:133–143, 1996.

[24] Danielle Pretolani. A directed hypergraph model for random time dependent shortest paths. *EJOR*, 123:315–324, sep 2000.

[25] A. J. V. Skriver. A classification af bicriteria shortest path (bsp) algorithms. Technical report, Department of Operations Research - University of Aarhus, Sep 1999. http://www.imf.au.dk/ ajs/bisurvey.ps.

[26] A. J. V. Skriver and K. A. Andersen. A label correcting approach for solving bicriterion shortest path problems. Technical report, University of Aarhus - Department of operational research, sep 1999. http://www.imf.au.dk/ ajs/biart.ps.

[27] Ralph E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. Wiley Interscience. Wiley, 1986.

[28] Robert Endre Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Conference Series*. SIAM, 1983.

[29] K. Thulasiraman and M.N.S. Swamy. *Graphs: Theory and Algorithms*. Wiley - Interscience, 1992.

[30] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

# Parametric shortest hyperpath problems

Lars Relund Nielsen* Kim Allan Andersen
Department of Operations Research
University of Aarhus
Ny Munkegade, building 530
DK-8000 Aarhus C
Denmark

**Abstract**

In this paper we consider the parametric shortest hyperpath problem in $B$-hypergraphs. We assume that the lengths of the hyperarcs depend on some parameter. We present procedures which find the shortest hyperpath from a source node to all other nodes in a $B$-hypergraph as a function of the parameter. The method extend results known from parametric shortest path problems in directed graphs.

Keywords: $B$-hypergraphs, Shortest hyperpaths, parametric shortest hyperpaths.

## 1 Introduction

In this paper we consider the parametric shortest hyperpath problem in $B$-hypergraphs. It is assumed that the lengths of the hyperarcs in the $B$-hypergraph depend on some parameter $\lambda$. We present procedures for determining the shortest hyperpaths from a source node $s$ to all other nodes $u$ hyperconnected to $s$ as a function of the parameter $\lambda$. To the best of our knowledge this problem has not been considered earlier.

The problem of determining the parametric shortest paths from a particular node $s$ to all other nodes $u$ in directed graphs has been presented in a number of papers. Probably the best known paper is Young, Tarjant, and Orlin [6]. However, the problem presented in that paper is somewhat simpler than the one presented in this paper. One reason is, that it only concerns digraphs, and another reason is, that there are some restrictions in the way, the parameter $\lambda$ is allowed to vary. In this paper, $\lambda$ is allowed to vary freely. This makes our problem more complicated, but probably also more oriented towards applications.

The paper is organized as follows. Section 2 contains preliminaries and Section 3 presents the problem along with a solution procedure. The section contains illustrating examples to make the presentation more clear. Section 4 list some items missing from the present version of this paper, which should be considered before the paper is published.

## 2 Preliminaries

This section contains the basic definitions of a directed B-hypergraph, i.e. hypergraphs where each hyperarc only have one node in its head. More general hypergraphs are presented in Gallo, Longo, Pallottino, and Nguyen [2]. In the following a B-hypergraph is referred to as a hypergraph.

*Corresponding author (e-mail: relund@imf.au.dk)

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

$$e = (T(e), h(e)) \qquad T(e) \subset \mathcal{V} \quad h(e) \subseteq \mathcal{V} \backslash T(e)$$

where $T(e)$ and $h(e)$ denote the *tail* nodes and the *head* node, respectively. The *cardinality* of hyperarc $e$ is the sum of the tail and head nodes, i.e.

$$|e| = |T(e)| + |h(e)| = |T(e)| + 1$$

If $|e| = 2$ hyperarc $e$ is called an *arc*. The *size* of $\mathcal{H}$ is the sum of the cardinalities of its hyperarcs:

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

We denote by

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

the *forward star* and the *backward star* of node $u$, respectively. A *path* $P_{st}$ in a hypergraph $\mathcal{H}$ is a sequence of nodes and hyperarcs in $\mathcal{H}$:

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

where, for $i = 1, ...q + 1, v_i \in T(e_i)$ and $v_{i+1} \in h(e_i)$. 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 $t \in T(e_1)$. A path is *cycle-free* if it does not contain any subpath which is a cycle, i.e.

$$v_i \in T(e_j) \Rightarrow j \geq i \qquad 1 \leq i \leq q + 1$$

If $\mathcal{H}$ contains no cycles, it is *acyclic*.

## 2.1 Ordering a hypergraph

We consider a topological ordering of the nodes of a hypergraph.

**Definition 1** Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A *valid ordering*

$$V = (v_{i_1}, v_{i_2}, ..., v_{i_n})$$

of the nodes in $\mathcal{H}$ is a topological ordering of the nodes, such that, for each $e \in \mathcal{E}$ and $u \in T(e)$, node $u$ precedes node $h(e)$ in the ordering (see Figure 1).

It is well-known that $\mathcal{H}$ is acyclic if and only if a valid ordering of the nodes in $\mathcal{H}$ is possible, see e.g. Gallo et al. [2] where a procedure also is presented.

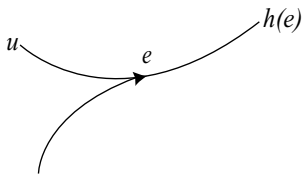



Figure 1: A valid ordering $V = (..., u, ..., h(e), ...)$.

## 2.2 Hyperpaths

We here use a slightly different definition of a hyperpath than in Gallo et al. [2] since, in some cases, this definition seems to be not working for B-hypergraphs, see Nielsen, Pretolani, and Andersen [4].
Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. A *hyperpath* $\pi_{st}$ of *origin* $s$ and *destination* $t$, is an acyclic minimal hypergraph (with respect to deletion of nodes and hyperarcs) $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ satisfying the following conditions:

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $s, t \in \mathcal{V}_\pi = \cup_{e \in \mathcal{E}_\pi} (T(e) \cup h(e))$
3. $u \in \mathcal{V}_\pi \setminus \{s\} \Rightarrow u$ is connected to $s$ in $\mathcal{H}_\pi$.

Note that 3. implies that for each $u \in \mathcal{V}_\pi \setminus \{s\}$ there exists a hyperarc $e \in \mathcal{E}_\pi$ such that $h(e) = u$, it follows from the minimality that $e$ is unique. Hyperarc $e$ is called the *predecessor* of $u$ and denoted by $e_\pi(u)$. Note that only a subhypergraph of $\mathcal{H}$ has to be considered when we want to find a hyperpath $\pi_{st}$ because the minimality also implies that the following condition holds:

4. $\exists\ u - t\ path \qquad \forall u \in \mathcal{V}_\pi \setminus \{t\}$

Therefore all nodes which do not have a path to $t$ can be removed from $\mathcal{H}$. We say that node $t$ is *hyperconnected* to $s$ if there exists a hyperpath $\pi_{st}$. Hyperpath $\pi_{st}$ is *different* from hyperpath $\pi_{uv}$ if they do not have the same hyperarcs.

## 2.3 Hypertrees

A *directed hypertree* with *root* $s$ is a hypergraph $\mathcal{T}_s = (\{s\} \cup \mathcal{N}, \mathcal{E}_\mathcal{T})$[1] satisfying the following conditions:

1. $\mathcal{T}_s$ is acyclic
2. $\{s\} \cap \mathcal{N} = \emptyset$
3. $BS(s) = \emptyset$
4. $|BS(v)| = 1 \quad \forall v \in \mathcal{N}$

A hypertree is the union of hyperpaths to all nodes in $\mathcal{N}$. If $FS(v) = \emptyset$ then $v$ is called a $leaf$. Note that different hypertrees can have the same hyperpath $\pi_{st}$.

## 2.4 Weighted hypergraphs

A *weighted hypergraph* is a hypergraph where each hyperarc $e$ is assigned a real weight $w(e)$. Given a hyperpath $\pi_{st}$ a weighting function $W_\pi$ is a node function which assigns weights $W_\pi(u)$ to all nodes in $\pi_{st}$. The weight of hyperpath $\pi_{st}$ is $W_\pi(t)$. We shall restrict ourselves to *additive weighting functions* which are defined by the recursive equations

$$W_\pi(u) = \begin{cases} w(e_\pi(u)) + F(e_\pi(u)) & u \in \mathcal{V}_\pi \setminus \{s\} \\ 0 & u = s \end{cases}$$

where $F(e)$ is a nondecreasing function of the weights of the nodes in $T(e)$. Two particular weighting functions, namely the *distance* and the *value,* have been studied in detail by Gallo et al. [2], and Pretolani [5] who showed that these two functions have practical applications. The *distance* is obtained by defining $F(e)$ as follows:

[1] In some definitions it is possible to have more than one root [1].

**Precondition:** Given hypergraph $\mathcal{H}$ with nondecreasing cycles and nonnegative hyperarc weights, let $W(v_i)$ denote the minimal weight in node $v_i$, $F(e)$ the chosen additive weighting function, $Q$ a candidate set and let $p$ be a predecessor function. Moreover, let the counter $k_j$ for each hyperarc $e_j$ represent the number of nodes in $T(e_j)$ which have been removed from $Q$. Therefore we just update $h(e_j)$ when the weights of all nodes in $T(e_j)$ has been calculated.

**Initialization:** Set $W(v_i) = \infty \ \forall i \in \mathcal{V}$, $k_j = 0 \ \forall e \in \mathcal{E}$, $Q = \{s\}$ and $W(s) = 0$

```
1  while (Q ≠ ∅) do
2      select and remove u ∈ Q;
3      for (e_j ∈ FS(u)) do k_j := k_j + 1
4          if (k_j = |T(e_j)|) then v := h(e_j)
5              if (W(v) > w(e_j) + F(e_j)) then
6                  if (v ∉ Q) then
7                      Q := Q ∪ {v}
8                      if (W(v) < ∞) then
9                          for (e_h ∈ FS(v)) do  k_h := k_h − 1
10                     end if
11                 end if
12                 W(v) := w(e_j) + F(e_j), p(v) := e_j
13             end if
14         end if
15     end for
16 end while
```

**Procedure 1:** Shortest B-tree procedure (SBT)

$$F(e) = \max_{v \in T(e)} \{W_\pi(v)\}$$

and the *value* is obtained as follows:

$$F(e) = \sum_{v \in T(e)} a_e(v) W_\pi(v)$$

where $a_e(v)$ is a nonnegative multiplier defined for each hyperarc $e$ and node $v \in T(e)$ (see Figure 2). The distance (the value) of a hyperpath $\pi_{st}$ is the weight of the hyperpath $\pi_{st}$ with respect to the distance (the value) weighting function.

## 2.5 The shortest hyperpath problem

The *shortest hyperpath problem* $(SBT)$[2] consists in finding the minimum weight hyperpaths from an origin $s$ to all nodes in $\mathcal{H}$ hyperconnected to $s$. In general the problem is hard to solve but if the weighting function is additive, fast algorithms exist. We first define a nondecreasing cycle which ensures that no weight can be decreased through a cycle.

[2]Shortest B-hypertree.

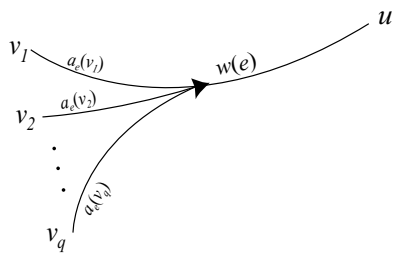



Figure 2: The weights and multipliers of the value function $e = (\{v_1, ..., v_q\}, \{u\})$.

**Precondition:** Given $V$, a valid ordering of $\mathcal{H}$, let $p$ denote the predecessor function and $F(e)$ the chosen additive weighting function.

**Initialization:** Set $W(s) := 0$, $W(v_i) = \infty$ $i = 1, ..., n$

```
1 for (i = 1 to n) do
2     for (e ∈ BS(v_i)) do
3         if (W(v_i) > w(e) + F(e)) then W(v_i) := w(e) + F(e), p(v_i) := e
4     end for
5 end for
```

**Procedure 2:** Shortest B-tree procedure when $\mathcal{H}$ is acyclic (SBT-acyclic)

**Definition 2** A *nondecreasing cycle* is a cycle $C = \{v_1, e_1, v_2, e_2, ..., v_r, e_r, v_1\}$ that satisfies

$$w(e_r) + F_{v_r}\left[w(e_{r-1}) + F_{v_{r-1}}(...F_{v_2}[w(e_1) + F_{v_1}(z)])\right] \geq z \qquad \forall z \in \mathbf{R}_+ \tag{2.1}$$

here $F_{v_i}(W)$ denotes the function where $v_i \in T(e_i)$ has weight $W$ and all other nodes $u \in T(e_i)$ has weight equal to zero.

That is, if node $v_1$ has temporary weight $z$ then going through $C$ will give no better temporary weight. Now, assume that the weighting function is additive, the weights nonnegative and that all cycles are nondecreasing. Gallo et al. [2] showed that finding the minimum weight hypertree is equivalent to finding a solution to *Bellmans generalized equations*

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

Procedure 1 proposed in [2] finds the minimum weight hypertree containing the shortest hyperpaths. If Dijkstra's principle is used, i.e. select from the candidate set $Q$ a node $u$ satisfying $W(u) = \min\{W(x) \mid x \in Q\}$ at each iteration, the well-known assumption of nonnegative arc weights in standard digraphs becomes

$$w(e) + F(e) \geq W(x) \qquad \forall x \in T(e),\ e \in \mathcal{E} \tag{2.2}$$

that is, the weight in the head must be equal to or larger than the weights in all the nodes in the tail. If assumption (2.2) is satisfied, then Dijkstra's theorem can be extended to hypergraphs.

**Theorem 1** Suppose assumption (2.2) is satisfied and $W(u) = \min\{W(x) \mid x \in Q\}$. 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 and hence line 9 in Procedure 1 can be dropped. Note if Dijkstra's theorem is used, then the order we pick the nodes from $Q$ define a valid ordering of the minimal hypertree.

### 2.5.1 Acyclic case

If $\mathcal{H}$ is acyclic, a fast procedure exists [3]. The procedure needs a valid ordering

$$V = (v_0 = s, v_1, ..., v_n)$$

and is stated in Procedure 2. The procedure finds the minimal weight for a node $v$ in one iteration, since the valid ordering assures that the minimal weights in the predecessor nodes have been found.

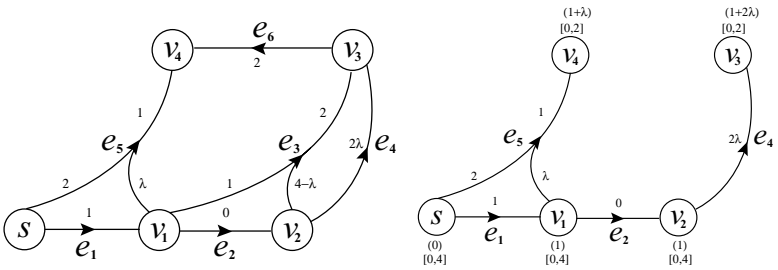
# 3 The parametric shortest hyperpath problem

Let $\mathcal{H}=(\mathcal{V},\mathcal{E})$ be a hypergraph where $\mathcal{V}=\{v_1,\ldots,v_n\}$ is the set of nodes and $\mathcal{E}=\{e_1,\ldots,e_m\}$ is the set of hyperarcs. Let $\Lambda=\left[\underline{\lambda},\overline{\lambda}\right]$ and assume that $\lambda\in\Lambda$. Assign to each hyperarc $e\in\mathcal{E}$ a nonnegative weight function $w_e(\lambda)$ and if the value function is considered nonnegative multiplier functions $a_e(u,\lambda)$ for $u\in T(e)$. That is, the weight and the multipliers are functions of the parameter $\lambda\in\Lambda$. We say that the hypergraph $\mathcal{H}$ is parametrized by the parameter $\lambda$ and denote it with $\mathcal{H}_\lambda$. Clearly, for a given value of $\lambda=\lambda_0$, we have that $\mathcal{H}_{\lambda_0}$ is a weighted hypergraph, i.e. weights and multipliers are fixed. Given a hyperpath $\pi$ we let $W_\pi(\lambda)$ denote the parametric weight of $\pi$.

Assume wlog that all nodes $v\in\mathcal{V}$ is hyperconnected to node $s$. In this section we want to determine a shortest hypertree from some source node $s$, i.e. the shortest hyperpaths from node $s$ to all other nodes, as a function of the parameter $\lambda\in\Lambda$. We call this problem the parametric shortest hyperpath problem.

Before presenting the solution procedure we shall illustrate it with a minor example.

**Example 1** A simple parametric hypergraph $\mathcal{H}_\lambda$ is shown in Figure 3(a). We consider the value weighting function where the weights and multipliers are written like in Figure 2.




(a) Hypergraph $\mathcal{H}_\lambda$ (b) $\mathcal{T}_s$ for $\mathcal{H}_0$

Figure 3: A simple parametric weighted hypergraph $\mathcal{H}_\lambda$ with $\Lambda=[0,4]$ and its minimal hypertree $\mathcal{T}_s$ for $\lambda=0$.

Now assume that $\Lambda=[0,4]$. The minimal weight hypertree for $\mathcal{H}_0$ is shown in Figure 3(b). We now want to find intervals of the parameter $\lambda$ so that the hypertree in Figure 3(b) is the optimal (minimal weight) one. By definition a shortest hyperpath from node $s$ to itself has value 0, and is optimal for all $\lambda\in\Lambda$. Also, the subpaths to nodes $v_1$ and $v_2$ are optimal for all values of $\lambda\in\Lambda$ since there is only one hyperpath to nodes 1 and 2. Moreover, the parametric weight of node $v_3$, when $e_4$ is used as a predecessor, is minimal if the parametric weight of node $v_3$, when another hyperarc is used as a predecessor is higher, i.e.

$$1+2\lambda\leq 1+(4-\lambda)+2\Leftrightarrow\lambda\leq 2$$

Hence it follows, that for $\lambda\in[0,2]$ edge $e_4$ is used as a predecessor in a shortest hyperpath to node $v_3$. Conversely if $\lambda\in[2,4]$ hyperarc $e_3$ is used as a predecessor in a shortest hyperpath to node $v_3$. For node $v_4$ the result are similar, however, we have to consider the intervals $\lambda\in[0,2]$ and $\lambda\in[2,4]$ separately, since one hyperarc in $BS(v_4)$ has and tail node in node $v_3$ and the predecessor of node $v_3$ is not the same in the two intervals. If $\lambda\in[0,2]$ we have

$$2\cdot 0+1\cdot\lambda+1\leq 1\cdot(1+2\cdot\lambda)+2\Leftrightarrow\lambda\geq -2$$

i.e. that for $\lambda \in [0,2]$ edge $e_5$ is used as a predecessor in a shortest hyperpath to node $v_4$. If $\lambda \in [2,4]$ we have

$$2 \cdot 0 + 1 \cdot \lambda + 1 \leq 1 \cdot (7 - \lambda) + 2 \Leftrightarrow \lambda \leq 4$$

and hence for $\lambda \in [2,4]$ edge $e_5$ is used as a predecessor in a shortest hyperpath to node $v_4$. We now have calculated a shortest hyperpath from node $s$ to node $v_i$, for $i = 1, \ldots, 4$ and for all $\lambda \in [0,4]$, as shown in Table 1.

| Node | $\lambda$ | $W_v(\lambda)$ | $p_v(\lambda)$ |
|---|---|---|---|
| 1 | [0,4] | 1 | $e_1$ |
| 2 | [0,4] | 1 | $e_2$ |
| 3 | [0,2] | $1 + 2\lambda$ | $e_4$ |
| | [2,4] | $7 - \lambda$ | $e_3$ |
| 4 | [0,4] | $1 + \lambda$ | $e_5$ |

Table 1: The hyperpath for $\lambda \in \Lambda$.

In conclusion the hypertree in Figure 3(b) is the minimal weight hypertree for $\lambda \in [0,2]$. ■

Next let us introduce some notation. Consider a particular node $u \in \mathcal{V}$ hyperconnected to $s$ and let $\pi_{su}$ be a shortest hyperpath from node $s$ to node $u$. We want to determine a set $\Lambda_u$ of values $\lambda$ for which $\pi_{su}$ is a shortest hyperpath to node $u$, i.e.

$$\lambda \in \Lambda_u \Rightarrow \pi_{su} \text{ is a shortest hyperpath for } \mathcal{H}_\lambda \tag{3.1}$$

This problem is denoted *the parametric shortest hyperpath problem* $(PSBT)$[3]. By definition $\Lambda_s = \Lambda$. Furthermore, the weight of node $s$ is always equal to 0, that is

$$W_s(\lambda) = 0 \quad \forall \lambda \in \Lambda_s = \Lambda \tag{3.2}$$

As it turns out, $\Lambda_u$ need not always has to be an interval, but $\Lambda_u$ can be split into disjoint intervals:

$$\Lambda_u \quad = [\underline{\lambda}_u^1, \overline{\lambda}_u^1] \cup [\underline{\lambda}_u^2, \overline{\lambda}_u^2] \cup \ldots \cup [\underline{\lambda}_u^q, \overline{\lambda}_u^q] = \Lambda_u^1 \cup \Lambda_u^2 \cup \ldots \cup \Lambda_u^q$$

Let the *backward node set* $BN_v$ of node $v$ denote the set of nodes $u$ which are in the tail of some arc $e \in BS(v)$, i.e.

$$BN_v = \{u \in \mathcal{V} \,|u \in T(e)\,,\ e \in BS(v)\}$$

Below we consider two cases, namely the general case and the acyclic case.

## 3.1 The general case

Consider $\mathcal{H}_\lambda = (\mathcal{V}, \mathcal{E})$ with $\lambda \in \Lambda$ and let $\lambda_0$ be a fixed number in $\Lambda$, we now have the following lemma.

**Lemma 1** Assume that $\mathcal{T}_s$ is the minimal hypertree for $\mathcal{H}_{\lambda_0}$ and that $\pi_{sv}$ is the corresponding shortest hyperpath to node $v$. Let $\Lambda_v$ denote the set

$$\Lambda_v = \Lambda_0 \cap \bigcap_{u \in BN_v} \Lambda_u$$

where

$$\Lambda_0 = \left\{\lambda \in \Lambda \mid w_{p(v)}(\lambda) + F_{p(v)}(\lambda) \leq w_e(\lambda) + F_e(\lambda)\,,\ \forall e \in BS(v) \setminus \{p(v)\}\right\}$$

Then $\Lambda_v$ satisfies condition (3.1).

[3]Parametric shortest B-tree.

**Precondition:** Let $\lambda_0$ denote a fixed number in $\Lambda$. Given a hypertree $\mathcal{T}_s$, let $W_v(\lambda)$ denote the weight of node $v$ and let $V_{\mathcal{T}_s} = (v_1, ..., v_n)$ denote a valid ordering of $\mathcal{T}_s$.

**Initialization:** Use an SBT procedure to find the minimal hypertree $\mathcal{T}_s$ of $\mathcal{H}_{\lambda_0}$.

```
1 for (i = 1 to n) do
2     Λ_{v_i} = ⋂_{u∈BN_{v_i}} Λ_u
3     for (e ∈ BS(v_i)) do
4         if (e ≠ p_{v_i}(λ_0)) then
5             Λ_0 = {λ ∈ Λ | W_{v_i}(λ) ≤ w_e(λ) + F_e(λ)}
6             Λ_{v_i} := Λ_{v_i} ∩ Λ_0
7         end if
8     end for
9 end for
```

**Procedure 3:** Finding values of $\lambda$ for which the minimal hypertree of $\mathcal{H}_{\lambda_0}$ is minimal.

**Proof** Follows immediately. ∎

Procedure 3 finds the minimum weight hypertree $\mathcal{T}_s$ for $\mathcal{H}_{\lambda_0}$, and calculate sets $\Lambda_v$ satisfying condition (3.1) for $v \in \mathcal{V}$. The procedure starts by using an SBT procedure to find a minimal hypertree $\mathcal{T}_s$ of $\mathcal{H}_{\lambda_0}$. Moreover, if Dijkstra's principle is used, then the order we pick the nodes in procedure SBT define a valid ordering of $\mathcal{T}_s$. Line 2 now finds the union of the intervals of the nodes in $BN_v$ and lines 5 and 6 the set from Lemma 1. Since the number of steps in Procedure 3 are finite, we have that Procedure 3 stops in a finite number of steps, provided that the set $\Lambda_0$ can be found in a finite number of steps.
Procedure 3 can also be used if some part of the minimal hypertree is known in advance. For instance suppose a minimal weight hypertree for the nodes $s, u_1, u_2, ..., u_{k-1}$ is known together with their corresponding sets $\Lambda_i$, satisfying condition (3.1). Then the for-loop in Procedure 3 can be changed from running from 1 to $n$, to run from $k$ to $n$ and only the rest of the minimal hypertree $\mathcal{T}_s$ has to be found in the initialization.
Note that Procedure 3 only finds sets $\Lambda_v$ for which the hypertree of $\mathcal{H}_{\lambda_0}$ is minimal. If we want to find a minimal hypertree for all $\lambda \in \Lambda$, we can use Procedure 3 as a subprocedure to solve this problem. We illustrate this idea by continuing Example 1.

**Example 1** (continued) Procedure 3 calculates the intervals $\Lambda_i = \left[\underline{\lambda}_i, \bar{\lambda}_i\right]$ for $i = 1, ..., 4$ and the minimal weight hypertree for $\mathcal{H}_{\lambda_0}$ (see Figure 3(b)). Now define $\Phi$ to be the set of all upper endpoints $(\bar{\lambda}_i)$. In this particular example we have

$$\Phi = \{2, 4\}$$

Assume we pick the minimal number, i.e. 2. We know that the hyperpaths to nodes 1 and

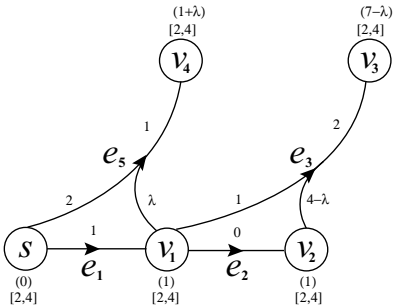



Figure 4: The minimal weight hypertree for $\mathcal{H}_2$ with $\Lambda = [2, 4]$.

**Precondition:** Assume that each call of Procedure 3 calculates a valid ordering $V_{\mathcal{T}_s} = (v_1, ..., v_n)$ of $\mathcal{T}_s$, weights $W_v(\lambda)$, predecessor hyperarcs and the sets $\Lambda_i$ which can be split into intervals

$$\Lambda_i = \Lambda_i^1 \cup \Lambda_i^2 \cup ... \cup \Lambda_i^{q_i} \quad \text{where } \Lambda_i^j = \left[\underline{\lambda}_i^j, \overline{\lambda}_i^j\right] \quad j = 1, ..., q_i$$

**Initialization:** Set $k = 1$, $\lambda_0 = \underline{\lambda}$ and $\bar{\Lambda} = \Lambda$.

1 **while** $(\lambda_0 \neq \overline{\lambda})$ **do** call Procedure 3 with $\lambda_0$, $\bar{\Lambda}$ and $k$ and set $\lambda_{old} := \lambda_0$
2 $\quad \Phi = \Phi \cup \left\{\overline{\lambda}_i^j\right\}$ for $i = k, ..., n$
3 $\quad$ select and remove $\lambda_0 = \min \Phi$
4 $\quad k := \min \left\{i \mid \exists \overline{\lambda}_i^j = \lambda_0\right\}$
5 $\quad \Lambda := \Lambda \setminus [\lambda_{old}, \lambda_0[$
6 **end while**

**Procedure 4:** Finding a minimal hyperpath to all nodes for each $\lambda \in \Lambda$.

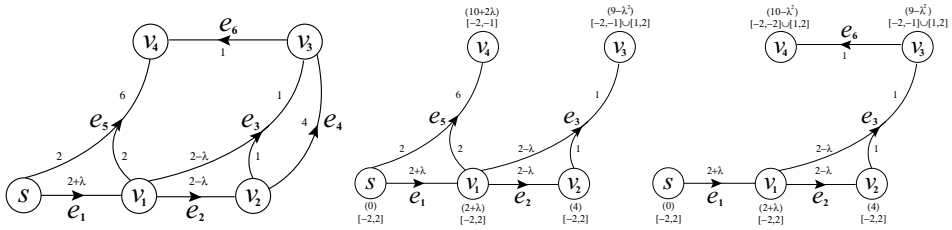
2 in Figure 3(b) are minimal for $\lambda = 2$, because 2 is contained in the intervals $\Lambda_1$ and $\Lambda_2$. $\lambda = 2$ is also contained in $\Lambda_3$, however, $\lambda = 2$ is an endpoint. This means that there is another hyperarc which give the same weight of node $v_3$ if $\lambda = 2$ (in this case hyperarc $e_3$). Now if we call Procedure 3 with $\lambda_0 = 2$, $k = 3$ and $\Lambda = [2, 4]$, we get the hypertree and intervals shown in Figure 4. We see that the hyperpath $\pi_{sv_4}$ is the same as before so hyperpath $\pi_{sv_4}$ is minimal for $\lambda \in [0, 4]$. We now could remove 2 from $\Phi$, add all new upper endpoints to $\Phi$, pick the minimal number and repeat the step with $\lambda_0 = 4$. However, since $\lambda_0 = 4 = \bar{\lambda}$ it is not necessary to repeat the step. We have now found a shortest hyperpath from node $s$ to node $v_i$ for $i = 1, ...4$ for all $\lambda \in [0, 4]$ (see Table 1).
Note that we have only found one (of possible many) shortest hyperpath for each $\lambda \in [0, 4]$, e.g. if $\lambda = 4$ the hyperpath $\pi_{sv_4}$ in Figure 4 is minimal but the hyperpath $\pi_{sv_4}$ with $p_{v_4}(4) = e_6$ instead of $p_{v_4}(4) = e_5$ is also minimal. ■

The procedure, demonstrated in the example above, is stated in Procedure 4 which finds a shortest hyperpath from node $s$ to node $v$ for all $\lambda \in \Lambda$. Here, we add the upper endpoint of $\Lambda_v^1$, for each $v \in \mathcal{V}$, to $\Phi$ on line 2. Note only the endpoint $\bar{\lambda}_v^1$ of the first interval $\Lambda_v^1$ in each node is added to $\Phi$. Next, we pick and remove the minimal endpoint on line 3, and finally, we calculate the new interval used in Procedure 3 on line 5. Note that, when Procedure 3 is called on line 1, the node $v_k$ satisfies that there is more than one hyperarc which can be used as a predecessor for node $v_k$. For instance in Example 1 we have that both hyperpaths $\pi_{sv_3}$ in Figure 3(b) and Figure 4 are minimal for $\lambda = 2$. However, if we use hyperarc $e_4$ as predecessor the interval $\Lambda_3$ calculated by Procedure 3 becomes $[2, 2]$. As the next example illustrates it is best to pick the predecessor hyperarc which gives the largest interval $\Lambda_v^1$.

**Example 2** Consider the hypergraph in Figure 5(a) where $\Lambda = [-2, 2]$ and the value is used as weighting function. When we call Procedure 4, we first run Procedure 3 with $k = 1$, $\Lambda = [-2, 2]$ and $\lambda_0 = -2$. Because there is only one hyperpath to nodes 1 and 2, we have the results stated in Table 2. For node 3, hyperarc $e_3$ used as predecessor gives the lowest weight, and we have to solve

| Node | $\lambda \in$ | $W_v(\lambda)$ | $p_v(\lambda)$ |
|---|---|---|---|
| 1 | $[-2, 2]$ | $2 + \lambda$ | $e_1$ |
| 2 | $[-2, 2]$ | 4 | $e_2$ |

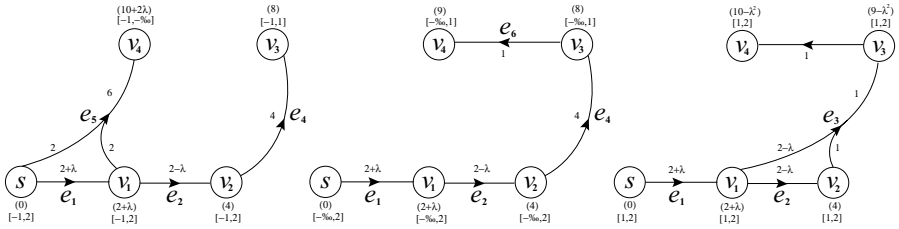
Table 2: The predecessor hyperarcs for $\lambda \in [-2, 2]$ when node 1 and 2 are considered.



(a) A parametric hypergraph.

(b) The hypertree for $\mathcal{H}_{-2}$ with $p_{v_4}(-2) = 5$.

(c) The hypertree for $\mathcal{H}_{-2}$ with $p_{v_4}(-2) = 6$.



(d) The hypertree for $\mathcal{H}_{-1}$.

(e) The hypertree for $\mathcal{H}_{-\frac{1}{2}}$.

(f) The hypertree for $\mathcal{H}_1$.

Figure 5: A parametric hypergraph and its corresponding minimal hypertrees.

$$
\begin{aligned}
&(2+\lambda)(2-\lambda) + 4 \cdot 1 + 1 \leq 4 + 4 \\
&\Rightarrow \Lambda_0 = [-\infty, -1] \cup [1, \infty] \\
&\Rightarrow \Lambda_3 = [-2, -1] \cup [1, 2]
\end{aligned}
$$

At node 4, we have that the weight $W_{v_4}(-2)$ using $e_5$ as predecessor hyperarc is equal to the weight using $e_6$ as predecessor hyperarc, so which hyperarc we pick as predecessor hyperarc depends on which rule we use to pick the predecessor hyperarc in Procedure 3. Assume that we pick $e_5$ first. Then we have to solve

$$
\begin{aligned}
&2(2+\lambda) + 6 \leq 9 - \lambda^2 + 1 \\
&\Rightarrow \Lambda_0 = [-2, 0] \\
&\Rightarrow \Lambda_4 = [-2, -1]
\end{aligned}
$$

and we get the hypertree shown in Figure 5(b). On the other hand, if we first pick $e_6$ we get the opposite

$$\Lambda_4 = [-2, -2] \cup [-1, 2]$$

This hypertree is shown in Figure 5(c). Suppose that the hypertree in Figure 5(c) was picked. Then an upper endpoint $\overline{\lambda}_4^1 = -2$ where added to $\Phi$ on line 2 and picked again on line 3 making $\Lambda$ on line 5 unchanged. The step is now repeated with $\lambda_0 = -2$, and if we do not have a rule saying that $e_6$ not must be picked as predecessor again, the procedure may loop. Assume that hyperarc $e_5$ is picked as predecessor, we then get the hypertree in Figure 5(b). $\Phi$ now becomes:

$$\Phi = \{-1, 2\}$$

Next Procedure 4 sets $\lambda_{old} = -2$, $\lambda_0 = -1$, $k = 3$ and $\Lambda = [-1, 2]$ and call Procedure 3. Assume that we pick $e_4$ as predecessor hyperarc for $v_3$ instead of $e_3$. The hypertree for $\mathcal{H}_{-1}$ is shown in Figure 5(d). Now -$\frac{1}{2}$ is added to $\Phi$ and Procedure 3 is called with $\lambda_0 = -\frac{1}{2}$, $k = 4$ and $\Lambda = \left[-\frac{1}{2}, 2\right]$. The hypertree is shown in Figure 5(e). Finally, Procedure 3 is called

| *node* | $\lambda \in$ | $W_v(\lambda)$ | $p_v(\lambda)$ |
|---|---|---|---|
| 3 | $[-2,-1]$ | $9-\lambda^2$ | $e_3$ |
| | $[-1,1]$ | $8$ | $e_4$ |
| | $[1,2]$ | $9-\lambda^2$ | $e_3$ |
| 4 | $\left[-2,-\frac{1}{2}\right]$ | $10+2\lambda$ | $e_5$ |
| | $\left[-\frac{1}{2},1\right]$ | $9$ | $e_6$ |
| | $[1,2]$ | $10-\lambda^2$ | $e_6$ |

Table 3: The predecessor hyperarcs for $\lambda \in [-2,2]$ when node 3 and 4 are considered.

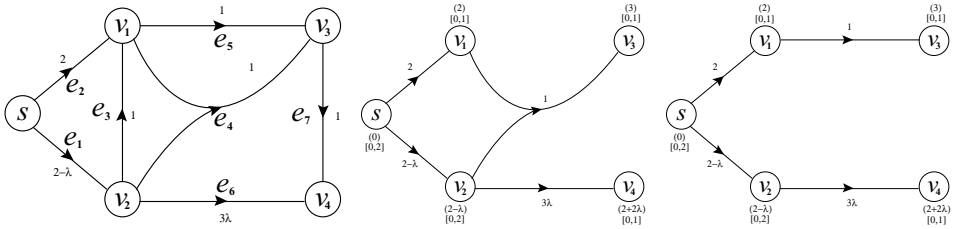
with $\lambda_0 = 1$, $k = 3$ and $\Lambda = [1,2]$ which gives the hypertree in Figure 5(f). This hypertree is actually identical to the hypertree in Figure 5(c) because the intervals in Figure 5(f) is a part of the intervals in Figure 5(c). We now have a minimal hyperpath $\forall\lambda \in [-2,2]$ to all nodes in $\mathcal{H}_\lambda$ (see Tables 2 and 3). ■

Let $Eq(v)$ be the set of possible predecessor hyperarcs which give minimal weight $W_v(\lambda_0)$ when hypergraph $\mathcal{H}_{\lambda_0}$ is considered. The above example suggests, that a possible rule for finding a minimal hypertree in Procedure 3 is:

Given set $\Lambda$ and hypergraph $\mathcal{H}_{\lambda_0}$, use hyperarc $e \in Eq(v)$ as predecessor for node $v$ in hyperpath $\pi_{sv}$, if no other hyperarc in $Eq(v)$ give a higher interval $\Lambda_v^1$.

By using this rule we would find the hypertrees in Figure 5 in 4 iterations.

All procedures described so far finds one (of possible many) minimal hyperpath to a node $v$, they do not find all minimal hyperpaths to a node $v$. This may be a problem when doing




(a) A parametric hypergraph.

(b) The hypertree for $\mathcal{H}_0$ with $p_{v_3}(0) = 4$.

(c) The hypertree for $\mathcal{H}_0$ with $p_{v_3}(0) = 5$.

(d) The hypertrees for $\mathcal{H}_1$ with $p_{v_4}(1) = 6$.

(e) The hypertrees for $\mathcal{H}_1$ with $p_{v_4}(1) = 7$.

Figure 6: A hypergraph $\mathcal{H}_\lambda$ and its corresponding minimal hypertrees.

sensitivity analysis, since it is very likely that the decision maker want to know if there are alternative hyperpaths that gives the same minimal weight. Then a representative set of minimal hyperpaths can be presented to the decision maker from which one can be chosen. However, all minimal hyperpaths can be found, if we for each endpoint in $\Phi$ and hyperarc $e \in Eq(v)$ calculate the minimal hypertree with $e$ as predecessor in node $v$. This can be illustrated with the following example.

**Example 3** Assume that if there exists more than one hyperarc in $Eq(v)$, then Procedure 3 calculate a minimal hypertree and sets $\Lambda_v$, satisfying condition (3.1) for each $e \in Eq(v)$. Now consider the parametric hypergraph in Figure 6(a) with $\Lambda = [0, 2]$. If we consider the distance function and call Procedure 3 with $\Lambda = [0, 2]$ and $\lambda_0 = 0$, we get the hypertrees in Figures 6(b) and 6(c). At nodes 1 and 2 there is only one predecessor hyperarc which is minimal for $\lambda_0$. At node 3 there are two possible predecessor hyperarcs which are both minimal for $\lambda_0$. Hence Procedure 3 calculate a hypertree for each predecessor which gives the figures in 6(b) and 6(c). We now have found all minimal hyperpaths for $\mathcal{H}_0$ and the intervals where they are minimal. If we now for each of the hypertrees call Procedure 3 with $k = 1$, $\lambda_0 = 1$ and $\Lambda = [1, 2]$, we get the hypertrees in Figures 6(d) and 6(e). Here two trees are shown in one figure because no matter the choice of predecessor in node $v_3$, the predecessor hyperarc for node 4 is the same. The figures show, for instance, that there are two minimal hyperpaths $\pi_{sv_3}$ which both are minimal for $\lambda \in [0, 2]$. ■

### 3.2 The acyclic case

If the hypergraph is acyclic the above procedures can be modified. Let $\mathcal{H}_\lambda = (\mathcal{V}, \mathcal{E})$ be an acyclic parametric weighted hypergraph with $\lambda \in \Lambda$. Because $\mathcal{H}_\lambda$ is acyclic a valid ordering $V$ of $\mathcal{H}_\lambda$ exists

$$V = (v_0 = s, v_1, ..., v_n)$$

This valid ordering can be used in Procedure 3. Furthermore, an acyclic SBT procedure can be called which use the valid ordering $V$, and hence becomes much faster since no heap is used to sort the nodes in the candidate set.

## 4 Future research

There is a number of problems which should be considered before this paper is finished. Below we list a few (of possibly many) missing items.

- In the introduction, we need a discussion of earlier work on parametric shortest paths. So far we have only made a link to the paper by Young et al. [6].
- We should discuss complexity issues in some more depth for the various procedures.
- Parts of the paper should be explained more carefully. In particular, the explanation concerning, if some part of the hypertree is known in advance, should be developed more carefully.
- A section on applications may be written. Here the procedures could be used on the random time-dependent shortest path problem, which can be transformed to a shortest hyperpath problem. If e.g. time and cost on some hyperarcs were not known for certain, a parameter $\lambda$ could be introduced on these hyperarcs.
- If we assume that only linear weights on each hyperarc is allowed, the weights in each node becomes linear. As a result the comparison of weights in each node becomes simple. This may be implemented in C++ and computational results carried out.

# References


[1] R. Cambini, G Gallo, and M. G. Scutella. Flows om hypergraphs. *Mathematical Programming*, 78:195–217, 1997.

[2] G. Gallo, G. Longo, S. Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied Mathematics*, 42:177–201, 1993.

[3] G. Gallo and S. Pallottino. Hypergraph models and algorithms for the assembly problem. Technical Report 6, Dipartimento di Informatica, March 1992.

[4] Lars Relund Nielsen, Daniele Pretolani, and Kim Allan Andersen. A remark on the definition of a B-hyperpath. Technical report, Department of Operations Research, University of Aarhus, 2001. Available at http://www.imf.au.dk/∼relund/.

[5] Danielle Pretolani. A directed hypergraph model for random time dependent shortest paths. *EJOR*, 123:315–324, sep 2000.

[6] Neal E. Young, Robert E. Tarjant, and James B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21:205–221, 1991.

# A remark on the definition of a B-hyperpath

Lars Relund Nielsen * Kim Allan Andersen
Department of Operations Research
University of Aarhus
Ny Munkegade, building 530
DK-8000 Aarhus C
Denmark

Daniele Pretolani
Dipartimento di Matematica e Fisica
Universitá di Camerino
Via Madonna delle Carceri
I-62032 Camerino (MC)
Italy

**Abstract**

In this note we show that a commonly used hyperpath definition for a directed B-hypergraph is wrong. This is done by presenting a counter-example which fulfils the hyperpath definition but which is not a hyperpath.

Keywords: Directed hypergraphs, B-hypergraphs, B-hyperpaths, Shortest hyperpaths, Weighted hypergraphs.

## 1 Introduction

Definitions of hyperpaths for directed hypergraphs have been used in many articles in recent years. Gallo, Longo, Pallottino, and Nguyen [2] introduced the widely used definition of a hyperpath called a Backward-hyperpath or a B-hyperpath. It was based on the topological properties of the B-hypertree found by a minimum weighting procedure. But the definition seems to fail in some cases. We here present a counter-example where the B-hypergraph satisfies the definition but can never be found by the minimum weighting procedure and hence it is not a B-hyperpath. A new definition of a B-hyperpath is suggested. In this paper, we only consider the definition of a B-hyperpath in Gallo et al. [2], and restrict ourselves to B-hypergraphs. For more general hypergraphs and hyperpath definitions see Gallo et al. [2]. Note that even though a wrong definition of a B-hyperpath has been used in many papers this has not contributed to wrong conclusions. All authors have used the minimum weighting procedure to calculate shortest B-hyperpaths, and the definition of a B-hyperpath has primarily been used to indicate what the minimum weighting procedure found.

## 2 A hyperpath counter-example

A *directed B-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 B-hyperarcs. A B-hyperarc $e \in \mathcal{E}$ is a pair $e = (T(e), h(e))$

*Corresponding author. Email: relund@imf.au.dk

**Precondition:** Given B-hypergraph $\mathcal{H}$ with nondecreasing cycles and nonnegative hyperarc weights, let $W(v_i)$ denote the minimal weight in node $v_i$, $F(e)$ the chosen additive weighting function and let $p : \mathcal{V} \to \mathcal{E}$, with $p(v) \in BS(v)$, be a predecessor function.

**Initialization:** Set $W(v_i) = \infty \ \forall i \in \mathcal{V}$, $k_j = 0 \ \forall e \in \mathcal{E}$, $Q = \{s\}$ and $W(s) = 0$

**while** $(Q \neq \emptyset)$ **do**
  Select and remove $u \in Q$;
  **for** $(e_j \in FS(u))$ **do** $k_j := k_j + 1$
    **if** $(k_j = |T(e_j)|)$ **then** $v := h(e_j)$
      **if** $(W(v) > w(e_j) + F(e_j))$ **then**
        **if** $(v \notin Q)$ **then**
          $Q := Q \cup \{v\}$
          **if** $(W(v) < \infty)$ **then**
            **for** $(e_h \in FS(v))$ **do** $k_h := k_h - 1$
          **end if**
        **end if**
        $W(v) := w(e_j) + F(e_j)$, $p(v) := e_j$
      **end if**
    **end if**
  **end for**
**end while**

**Procedure 1:** Shortest B-tree procedure

with $T(e) \subset \mathcal{V}$ and $h(e) \subseteq \mathcal{V} \backslash T(e)$, where $T(e)$ and $h(e)$ denote the *tail* nodes and the *head* node, respectively.
We denote by $FS(u) = \{e \in \mathcal{E} \mid u \in T(e)\}$ and $BS(u) = \{e \in \mathcal{E} \mid u \in h(e)\}$ the *forward star* and the *backward star* of node $u$, respectively. A *path* $P_{st}$ in a B-hypergraph $\mathcal{H}$ is a sequence of nodes and B-hyperarcs in $\mathcal{H}$:

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

where, for $i = 1, ...q + 1, v_i \in T(e_i)$ and $v_{i+1} \in h(e_i)$. 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 $t \in T(e_1)$. A path is *cycle-free* if it does not contain any subpath which is a cycle, i.e. $v_i \in T(e_j) \Rightarrow j \geq i \quad 1 \leq i \leq q+1$. If $\mathcal{H}$ contains no cycles, it is *acyclic*. $\mathcal{H}$ is called a *weighted hypergraph* if each B-hyperarc $e \in \mathcal{E}$ is assigned a real weight $w(e)$.
For a hypergraph Gallo et al. [2] presented a definition of a B-hyperpath which has been used in many papers, see e.g. Gallo and Pallottino [3], Nguyen and Pretolani [6], Pretolani [8]. If in particular a B-hypergraph is considered the definition becomes:

**Definition 1 (Gallo et al. [2])** Given a B-hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, a *B-hyperpath* $\pi_{st}$ of *origin* $s$ and *destination* $t$, is a minimal B-hypergraph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ (with respect to deletion of nodes and hyperarcs) satisfying the following conditions:

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $s, t \in \mathcal{V}_\pi = \cup_{e \in \mathcal{E}_\pi} (T(e) \cup h(e))$
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 $P_{su}$.

Procedure 1 was suggested by Gallo et al. [2] to find a minimal B-hyperpath from $s$ to $t$. The procedure solves the shortest hyperpath problem (SBT) of a weighted B-hypergraph which consists in finding the minimum weight hyperpaths from an origin $s$ to all other nodes in $\mathcal{H}$. Procedure 1 finds a subhypergraph $\mathcal{T}$ of $\mathcal{H}$, defined by the predecessor function $p$. $\mathcal{T}$ is called the minimum weight B-hypertree. It is well-known that the minimal B-hypertree is the union of all minimal B-hyperpaths from $s$ to all other nodes $v$ in $\mathcal{H}$ for which a B-hyperpath exists ($W(v_i) < \infty$).

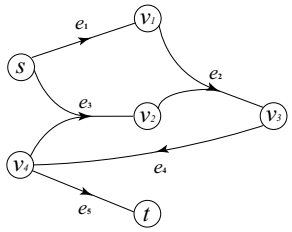



Figure 1: A hypergraph satisfying the definition of a hyperpath given in Gallo et al. (1993), which is not a hyperpath.

The B-hypergraph $\mathcal{H}$ in Figure 1 fulfils Definition 1. There is a cycle-free path from node $s$ to all other nodes $v$, e.g. $P_{sv_4} = (s, e_1, v_1, e_2, v_3, e_4, v_4)$ is cycle-free. But $\mathcal{H}$ is not a B-hyperpath. This can be seen if we set the weight of each hyperarc $e$ in Figure 1 to one and use the sum weighting function[1]. Procedure 1 will now examine node $s$ and $v_1$ and then stop. So $\mathcal{H}$ is not contained in the B-hypertree and hence not a B-hyperpath. Definition 1 fails because $|h(e)| = 1\ \forall e \in \mathcal{E}$ implies that only one hyperarc will enter a node $v$ in a B-hyperpath (minimality for B-hypergraphs). This implies that a B-hyperpath of a B-hypergraph is acyclic because otherwise the weight is improved through a cycle (only nondecreasing cycles). Condition 3 in Definition 1 fails to assure this for $\mathcal{H}$ in Figure 1 and is therefore not strong enough. A slightly changed definition of a hyperpath in a B-hypergraph is

**Definition 2** Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ denote the B-hypergraph considered. A *B-hyperpath* $\pi_{st}$ of *origin* $s$ and *destination* $t$, is an acyclic minimal B-hypergraph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ (with respect to deletion of nodes and hyperarcs) satisfying the following conditions:

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $s, t \in \mathcal{V}_\pi = \cup_{e \in \mathcal{E}_\pi} (T(e) \cup h(e))$
3. $u \in \mathcal{V}_\pi \setminus \{s\} \Rightarrow u$ is connected to $s$ in $\mathcal{H}_\pi$.

Here the minimality assures that only one hyperarc enters each node $v \in \mathcal{V}_\pi \setminus \{s\}$, and condition 3 that it is connected to $s$. Note that condition 3 in Definition 2 is less restrictive than condition 3 in Definition 1, however, the fact that the B-hyperpath has to be acyclic together with condition 3 in Definition 2 is more restrictive.

# References

[1] 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.

[2] G. Gallo, G. Longo, S. Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied Mathematics*, 42:177–201, 1993.

[3] G. Gallo and S. Pallottino. Hypergraph models and algorithms for the assembly problem. Technical Report 6, Dipartimento di Informatica, March 1992.

[4] G. Gallo and M. G. Scutella. Minimum makespan assembly plans. Technical Report 10, Dipartmento di Informatica - Universita di Pisa, 1998.

[1] $F(e) = \sum_{v \in T(e)} W(v)$

[5] R. G. Jeroslow, K. Martin, R. L. Rardin, and J. Wang. Gainfree leontief substitution flow problems. *Mathematical Programming*, 57:375–414, 1992.

[6] S. Nguyen and D. Pretolani. Shortest hyperpath problems on oriented hypergraphs. Technical report, Centre de recherche sur les transports, Sep 1994.

[7] S. Nguyen, D. Pretolani, and L. Markenzon. On some path problems on oriented hypergraphs. *RAIRO - Informatique theorique et appliqations*, (32 No. 1), 1998.

[8] Danielle Pretolani. A directed hypergraph model for random time dependent shortest paths. *EJOR*, 123:315–324, sep 2000.