PhD Thesis 2001

Department of Operations Research

University of Aarhus

Multicriteria Analysis on Network and Location Problems

Anders J.V. Skriver

Preface

In August 1997 I started working on the Bicriterion Shortest Path (BSP) problem. This was meant as an introduction to the field of multicriteria network problems. The BSP problem is one of the simple problems in the field, yet it is still $\mathcal{NP}$ complete. I wrote down comments on the related papers in the form of a short survey paper “A Classification of Bicriteria Shortest Path (BSP) algorithms” (Paper A). Together with my supervisor Kim Allan Andersen, we have found a preprocessing rule for the label correcting solution approach presented in “A label correcting approach for solving bicriterion shortest path problems” (Paper B).

In the late spring of 1998 I started to investigate the Bicriterion Semi-obnoxious Planar Location (BSPL) problem. This was inspired by the plans of building a new international airport near the city of Aarhus, Denmark. Together with Kim Allan Andersen, we have set up a bicriterion model for this problem, and adapted an approximate solution method called Big-Square-Small-Square (BSSS). In the spring of 1999 I started to apply the same method in the network model of the same location problem (BSNL), and we have presented both models in “The Bicriterion Semi-obnoxious Location (BSL) Problem Solved by an ϵ-Approximation” (Paper C).

In the late spring of 1999 I visited professor Horst Hamacher at the University of Kaiserslautern, Germany, for four months. During this period I have worked together with Prof. Hamacher on a general exact solution method for the multicriteria location problem on a network with both pull and push objective functions. The results are presented in “Multicriteria Semi-obnoxious Network Location (MSNL) Problems with Sum and Center Objectives” (Paper D).

In the early spring of 2000 professor Kaj Holmberg, Linköping Institute of Technology, Sweden, visited our department. Together with Kim Allan Andersen we started a research project on an extension of the MSNL problem, where the edge-lengths are made criteria dependent. The resulting problem is a mix of the BSP and the MSNL problems. The results are presented in “Bicriteria Network Location (BNL) problems with criteria dependent lengths and minisum objectives” (Paper E).

During the spring of 2000 I started an application oriented project together with Morten Riis, a PhD student at our department, and Jørn Lodahl, Sonofon. During several meetings at Sonofon, we formulated a two-stage stochastic programming model, to describe a capacity expansion problem. The results are presented in “Network planning in telecommunications: A stochastic programming approach” (Paper F).

In the late fall 2000, Kim Allan Andersen and I invited Matthias Ehrgott, University of Auckland, New Zealand, to visit our department. During his stay Matthias and I formulated an algorithm for the Max-ordering (MO) problem in a combinatorial context. The results are presented in "Solving Biobjective Combinatorial Max-Ordering Problems by Ranking Methods and a Two-Phases Approach" (Paper G).

Acknowledgments

Credits are due to Horst Hamacher, Matthias Ehrgott, Stefan Nickel, Kaj Holmberg, Jørn Lodahl, Morten Riis, Lars Relund Nielsen, Philip Melchiors and my supervisor Kim Allan Andersen.

Publication status

- Paper A: **A Classification of Bicriteria Shortest Path (BSP) Algorithms**, Asia-Pacific Journal of Operational Research 17, (2000), 199-212. A.J.V. Skriver.

- Paper B: **A label correcting approach for solving bicriterion shortest path problems**, Computers and Operations Research 27, (2000), 507-524. A.J.V. Skriver and K.A. Andersen.

- Paper C: **The Bicriterion Semi-obnoxious Location (BSL) Problem Solved by an ϵ-Approximation**, Submitted. A.J.V. Skriver and K.A. Andersen.

- Paper D: **Multicriteria Semi-obnoxious Network Location (MSNL) Problems with Sum and Center Objectives**, Submitted. H.W. Hamacher, M. Labbé, S. Nickel and A.J.V. Skriver.

- Paper E: **Bicriteria Network Location (BNL) problems with criteria dependent lengths and minisum objectives**, Submitted. A.J.V. Skriver, K.A. Andersen and K. Holmberg.

- Paper F: **Network planning in telecommunications: A stochastic programming approach**, Submitted. M. Riis, A.J.V. Skriver and J. Lodahl.

- Paper G: **Solving Biobjective Combinatorial Max-Ordering Problems by Ranking Methods and a Two-Phases Approach**, Submitted. M. Ehrgott and A.J.V. Skriver.

All seven papers are expected to be published in international, reviewed journals.

Contents

1 Introduction

Multicriteria analysis on networks is the main theme of this thesis, but I have also looked at two different problems. A planar location problem and a single objective network problem arising in mobile telecommunications.

Multicriteria analysis is focused on mathematical optimization problems with more than one objective. There is a general theory for the overall problem class, but the results are of course very general. I try to develop this theory further for problems in which some structural knowledge can be used to achieve a better solution procedure. Most of the problems are network problems, and as such they can be formulated as integer programming problems with more than one objective. Since most integer linear programming problems are $\mathcal{NP}$-complete, these problems are at least as hard. It should be mentioned that problems that are polynomially solvable with one objective, may be $\mathcal{NP}$-complete with two objectives. This is the case for the Shortest Path (SP) problem.

I will shortly describe the relevant problems, followed by an introduction to the basic concepts of multicriteria analysis. The first problem is the Bicriteria Shortest Path (BSP) problem described in more detail in Section 2. This obvious generalization of the traditional shortest path problem, in which one has to find the shortest (cheapest) path from a source node s to a terminal node t. In the BSP problem we simply have two objectives, namely time and cost. This model reveals the trade-off between the two objectives.

The second problem is the Max-ordering (MO) problem, examined in a combinatorial context. Here the objective is to minimize the maximum objective value. This problem arises as a subproblem in general multicriteria solution approaches such as the interactive weighted Tchebycheff method. The problem is described in detail in Section 3.

The third problem is the single facility location problem where different variants are described in Sections 4, 5 and 6. The problem is to locate one new facility in a scenario with a number of existing facilities. The new facility will of course interact with the existing facilities, and this interaction is assumed to depend on the distance between the new and the existing facilities. The way this interaction takes place is represented by the objective function(s). In the single objective case, this problem has been well studied, and the two most common objectives are the median (minimizing the sum of weighted distances) and the center (minimizing the maximum weighted distance). These two objectives represent a pull effect, meaning that the new facility is favored, consequently the distance should be minimized. If we consider an undesirable (obnoxious) facility the objective reflects that the distance between the existing facilities and the new facility should be maximized.

This is often referred to as a push effect. The two most obvious objectives for this problem is the anti-median (maximizing the sum of weighted distances) and the anti-center (maximizing the minimum weighted distance). These problems with different objectives have been examined both in the plane and on networks. In the plane there are different possibilities to measure the distance, the most popular being the l^p-norm. When a facility is both favored and obnoxious it is referred to as **semi-obnoxious**.

The final problem is somewhat different from the above-mentioned problems. The problem of expanding the capacity of a mobile communications network, modeled by a two stage stochastic program, was inspired by a problem instance at Sonofon. The model is described in Section 7.

1.1 Terminology of multicriteria analysis

I will now introduce some concepts in multicriteria analysis. For a textbook introduction see Steuer [68] or Ehrgott [24]. Consider the following general multicriteria problem:

$$\begin{array}{ll} \min & f^1(x) \\ \min & f^2(x) \\ \vdots & \quad \vdots \\ \min & f^k(x) \\ \text{s.t.} & \\ & x \in S \end{array} \tag{1}$$

$S \in \mathbb{R}^n$ is the set of feasible solutions, and $f(x) = (f^1(x), \ldots, f^k(x))$. Solving the multicriteria problem means finding the optimal solution. But what is an optimal solution, when we have k objective functions instead of one? The answer is efficient solutions. A solution is called **efficient** (Pareto optimal) if we cannot improve one objective value without worsening another. The mathematical definition of efficiency is as follows.

Definition 1 *A point $x \in S$ is **efficient** iff there does not exist a point $\bar{x} \in S$ such that $f(\bar{x}) \leq f(x)$ with at least one strict inequality. Otherwise x is **inefficient**.*

Please note that efficient points are the same as Pareto optimal points. A less restrictive definition of efficient points, called weakly efficient points is defined as follows.

Definition 2 *A point $x \in S$ is **weakly efficient** iff there does not exist a point $\bar{x} \in S$ such that $f(\bar{x}) < f(x)$, i.e. $f^i(\bar{x}) < f^i(x)\ \forall i = 1, \ldots, k$.*

Efficient points are defined in decision space. There is a natural counterpart in criterion space, where the criterion space $\mathcal{Z}$ is defined as $\mathcal{Z} = \{z \in R^k | \exists x \in S, z = f(x)\}$. Thus the criterion vectors correspond to the image of a mapping of all the feasible solutions to (1).

Definition 3 *$z(x) \in \mathcal{Z}$ is a* ***nondominated*** *criterion vector iff x is an efficient solution. Otherwise $z(x)$ is a* ***dominated*** *criterion vector.*

In the above Definition 3 we have used that $z(x) = f(x)$. The set of efficient solutions are denoted $\mathcal{X}_{Par}$, and the set of nondominated criterion vectors are denoted $\mathcal{Z}_{Par}$, and is given by $\mathcal{Z}_{Par} = z(\mathcal{X}_{Par})$.
Since the decision maker's utility function is usually unknown (also to herself), a solution to (1) is to find all efficient solutions (or all nondominated vectors).
The criterion vectors can be partitioned into two kinds, supported and unsupported. The supported can then be further divided into supported extreme and supported non-extreme. Following the terminology of Steuer [68] we define $\mathcal{Z}^{\geq}$:

$$\mathcal{Z}^{\geq} = Conv(\mathcal{Z}_{Par} \oplus \{z \in R^k | z \geq 0\}) = Conv(\mathcal{Z}_{Par}) \oplus \{z \in R^k | z \geq 0\}$$

where $\oplus$ signifies set addition and $Conv$ means convex hull. From this set we can characterize the different criterion vectors.

Definition 4 *$z \in \mathcal{Z}_{Par}$ is a* ***supported*** *nondominated criterion vector if z is on the boundary of $\mathcal{Z}^{\geq}$. Otherwise z is an* ***unsupported*** *nondominated criterion vector.*

It is important to note that unsupported nondominated criterion vectors are dominated by a convex combination of other nondominated criterion vectors.

Definition 5 *$z \in \mathcal{Z}_{Par}$ is a* ***supported extreme*** *nondominated criterion vector if z is an extreme point of $\mathcal{Z}^{\geq}$.*

Among the supported nondominated criterion vectors the extreme vectors are the most important, because they can be found as extreme point solutions when minimizing a convex combination of the k objective functions. This is mainly interesting when the objective functions are linear, which is often the case. We define the objective function $W(x, \lambda)$ as follows:

$$W(x, \lambda) = \sum_{i=1}^{k} \lambda_i f^i(x), \ \lambda \in \Lambda \tag{2}$$

where $\Lambda = \{\lambda \in R^k | \lambda_i > 0, \sum_{i=1}^{k} \lambda_i = 1\}$. The function $W(x, \lambda)$ is a convex combination, or **weighted sum**, of the k objective functions. If S is a convex set and f^i are convex functions, then optimizing (2) with different λ vectors will give the supported (extreme) nondominated vectors (Geoffrion [34]). Therefore, it is often referred to as the **weighting**

method or the Parametric method. Because unsupported nondominated criterion vectors are dominated by a convex combination of supported nondominated criterion vectors, unsupported nondominated vectors cannot be found by the weighting method. This is illustrated in Figure 1. The solution(s) x in decision space corresponding to a supported (extreme) criterion vector can be referred to as a supported (extreme) solution.

It is often convenient to initially solve the problem with respect to the k objectives independently in order to find the respectively minimal values. Actually, it is often better to solve a slightly perturbed version of the k problems in order to avoid weakly efficient points. Assume ϵ is a very small strictly positive constant. Then solve

$$\min_{x \in S} f^i(x) + \epsilon \sum_{j \neq i} f^j(x) \quad \forall i = 1, \ldots, k \tag{3}$$

and denote the corresponding optimal solutions $x^1, \ldots, x^k$. Let $f^{i*} = f^i(x^i)\ \forall i$. The **pay-off table** in Table 1 then lists how the individually optimal solutions are located in relation to each other. The diagonal elements form the **ideal point** $f^* = (f^{1*}, \ldots, f^{k*})$.

	x^1	x^2	$\cdots$	x^k
f^1	f^{1*}	$f^1(x^2)$	$\cdots$	$f^1(x^k)$
f^2	$f^2(x^1)$	f^{2*}	$\cdots$	$f^2(x^k)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
f^k	$f^k(x^1)$	$f^k(x^2)$	$\cdots$	f^{k*}

Table 1: Pay-off table.

2 The Bicriterion Shortest Path (BSP) problem

The BSP problem is one of the simplest problems in multicriteria integer analysis, but nevertheless also one of great importance in many applications. One of them being transportation problems with more than one objective. Furthermore, the BSP problem often occurs as a subproblem in other problems like scheduling problems. It also occurs as a subproblem in models for transportation of hazardous materials, see Erkut *et al.* [29].

2.1 The model

Let us formulate the problem mathematically. We are given a strongly connected *directed network* or a *digraph* $G = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N} = \{1, \ldots, n\}$ is the set of nodes and $\mathcal{E} = \{(i,j), (k,l), \ldots, (p,q)\}$ is a finite set of directed edges (arcs) joining nodes in $\mathcal{N}$. Parallel edges are allowed. Each edge $(i,j) \in \mathcal{E}$ carries two attributes denoted by (c_{ij}, t_{ij}). Often these coefficients are assumed to be positive, but it is enough to require that no negative cycles exist. For simplicity assume that c_{ij} is the cost using edge (i,j) and t_{ij} is the travel time from node i to node j (using the edge (i,j)). The objective is to find the set of efficient paths from a particular node, the source node $s \in \mathcal{N}$, to another particular node, the terminal node $t \in \mathcal{N}$. Traditionally, the BSP problem is formulated as follows:

$$
\begin{array}{ll}
\min & c(x) = \sum_{(i,j)\in\mathcal{E}} c_{ij} x_{ij} \\
\min & t(x) = \sum_{(i,j)\in\mathcal{E}} t_{ij} x_{ij} \\
\text{s.t.} & \\
& \sum_{\{j|(i,j)\in\mathcal{E}\}} x_{ij} \; - \sum_{\{j|(j,i)\in\mathcal{E}\}} x_{ji} = \left\{ \begin{array}{rl} 1 & \text{if } i = s \\ 0 & \text{if } i \neq s,t \\ -1 & \text{if } i = t \end{array} \right. \\
& x_{ij} \in \{0,1\}, \;\; \forall \;\; (i,j) \in \mathcal{E}
\end{array}
\tag{4}
$$

The constraints in (4) yield a directed path from source node s to terminal node t and the two objectives are to find the minimum cost $s-t$ path and the minimum travel time $s-t$ path, respectively. The problem is known to be $\mathcal{NP}$-complete by transformation from a 0-1 knapsack problem, Garey and Johnson [33].

It is well-known that the constraint set in (4) defines an integral polytope (the constraint-matrix is totally unimodular). Therefore, if the linear relaxation of (4) is solved using the weighting method, the set of supported (extreme) efficient paths is found. Unfortunately there may be a lot of unsupported efficient paths such as D indicated in Figure 1.

Being interested in the set of efficient paths, it is not a satisfactory compromise just finding the set of supported efficient paths.

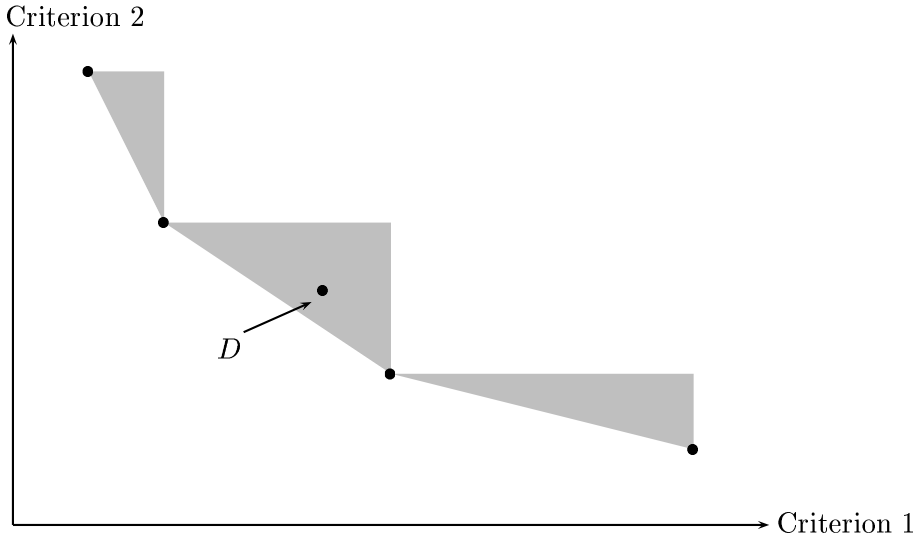

Figure 1: D is an unsupported, nondominated criterion vector.

2.2 Solution approaches

To our knowledge there are four survey papers including the BSP problem, Zionts [77], Rasmussen [63], Ulungu and Teghem [73] and Paper A [65]. The first two references survey the general multicriteria integer programming problem for which the BSP is a special case, and both papers are relatively old. The third reference surveys many of the papers also included in Paper A. The main contribution of Paper A is a classification of the existing solution methods, and a ranking of the methods based on the algorithmic structure. Ehrgott and Gandibleux [25] have recently written a bibliography paper on Multiobjective Combinatorial Problems (MOCO) containing more than 350 references, including also the BSP problem.

There are generally two main approaches, a path/tree approach and a node labeling approach, see Figure 2. Each of the two main approaches are again divided into two. The path/tree approach splits into the K'th shortest path approach and the Two-Phases method. The node labeling approach splits into a Label Setting and a Label Correcting approach.

In a path approach we examine different path vectors, and try to find the efficient ones. Similarly, we investigate the m dimensional incidence vectors that characterize the different spanning trees in a tree approach. Since there are usually many edges compared to the number of nodes and there may be exponentially many spanning trees, a labeling approach that compares values in the two-dimensional criterion space at each node may be advantageous. In a Label Setting approach one label is made permanent in each iteration

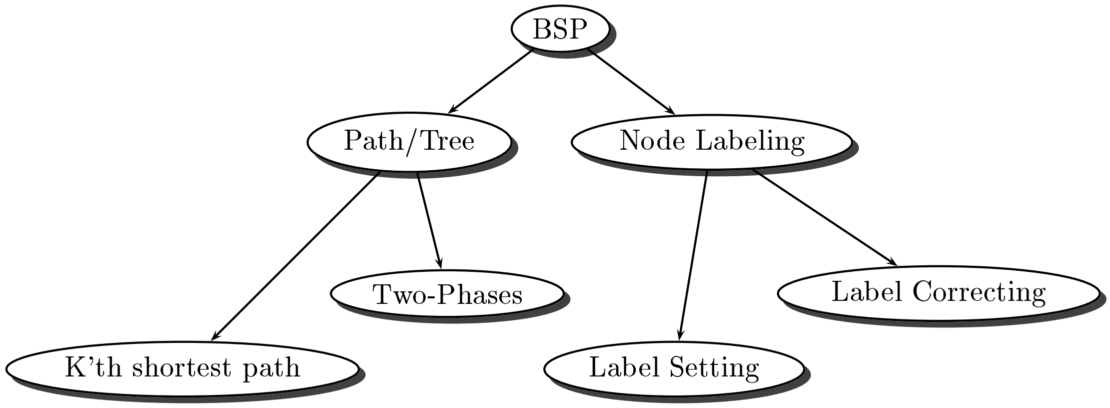

Figure 2: Classification of BSP algorithms.

and in a Label Correcting approach all labels are changeable until the stop criterion is fulfilled.

K'th shortest path	Climaco and Martins [16]
Two-Phases	Coutinho-Rodrigues, Climaco and Current [19] Mote, Murthy and Olson [55]
Label Setting	Hansen [40], Martins [50] algorithm 1, Tung and Chew [72]
Label Correcting	Brumbaugh-Smith and Shier [10], Corley and Moon [18], Daellenbach and DeKluever [20], Skriver and Andersen [66]

Table 2: Classification of references.

In Table 2 we list the references that fall in the four categories. The number of references applying a labeling approach indicates that this is the most successful approach. The second phase in Coutinho-Rodrigues, Climaco and Current [19] is actually an K'th shortest path approach, and the first phase solves the LP relaxation of (4). The second phase of Mote, Murthy and Olson [55] is a Label Correcting approach, and their first phase solves an LP relaxation of a spanning tree problem closely related to (4). In Paper A the four different approaches are discussed in more detail.

Next we illustrate the complexity of the BSP problem by a small example. We use the example to explain why the node-labeling approach is better than the path/tree handling procedure. For clarity remember that efficient paths are in the (high dimensional) decision space, and the nondominated values are in the (two-dimensional) criterion space. The example is similar to one found in Hansen [40], and is presented in more detail in Paper B [66].

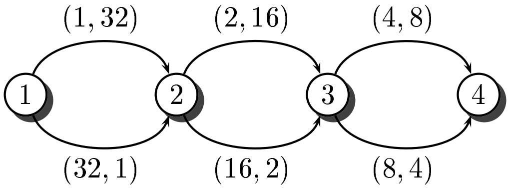

Figure 3: Example with exponentially many nondominated values.

Here we choose the edge coefficients, so that the sum of the smaller coefficients is less than that of the next. This is achieved by the following numbers, 2^i, $i = 0, 1, 2, \ldots, |\mathcal{E}| - 1$. In this example that is $1, 2, 4, 8, 16, 32$, which we then pair from each end of the list as shown in Figure 3. These power of two coefficients are often used in MOCO problems to illustrate that the problem is **intractable**, which means that the number of (different) efficient solutions may grow exponentially.

The 8 paths in the network in Figure 3 are all efficient having the following 8 nondominated values: $(7, 56)$, $(11, 52)$, $(21, 42)$, $(25, 38)$, $(38, 25)$, $(42, 21)$, $(52, 11)$ and $(56, 7)$. By choosing the edge coefficients this way we get $2^{|\mathcal{N}-1|}$ nondominated values.

From this special case of the BSP problem we make two observations. The number of efficient paths may grow exponentially in the number of nodes, namely $2^{|\mathcal{N}-1|}$, and the number of efficient paths is always greater than or equal to the number of nondominated values, because we may have paths with the same objective function values. The last observation can also be made from Definition 3, because $\mathcal{Z}_{Par} = z(\mathcal{X}_{Par})$. If all edge-weights are $(1, 1)$, there is only one nondominated value, namely $(3, 3)$, but all 8 paths are efficient.

2.3 The Brumbaugh-Smith and Shier algorithm

The algorithm below is taken directly from Brumbaugh-Smith *et al.* [10]. It is included to make the presentation self-contained, because the preprocessing rules in Section 2.4 are designed for this particular algorithm.

Let $D(i) = \{(c_1(i), t_1(i)), \cdots, (c_p(i), t_p(i))\}$ be the label-set at node i containing p labels. At each step these labels are nondominated by any other label in the set. The labels are sorted by increasing cost values. The set Labeled is a set of nodes that needs to be examined. The FIFO principle is used to select nodes from the set Labeled as recommended in Brumbaugh-Smith *et al.* [10]. By $out(i)$ we refer to the nodes j for which $(i, j) \in \mathcal{E}$. The merge operator of the sets A and B is defined as

$$Merge(A, B) = (A \cup B) \setminus \{z \in A \cup B \mid \exists\, x \in A \cup B : x \leq z\}$$

This means that after the sets are joined all dominated labels are deleted.

Algorithm 2.3:

1. Initialize:

   $D(s) = \{(0,0)\}$;

   Labeled=$\{s\}$;

2. while Labeled $\neq \emptyset$

   choose i from Labeled;

   Labeled=Labeled-$\{i\}$;

   for $j \in out(i)$

   $D_M(j) = Merge(D(j), D(i) + (c_{ij}, t_{ij}))$;

   If $D(j) \neq D_M(j)$ then

   $D(j) = D_M(j)$;

   If j is not in Labeled then (avoids double labeling)

   Labeled = Labeled + $\{j\}$;

In this algorithm the merge operation uses the main part of the computational effort. Our intention was to discard "expensive" edges before the merge operation is carried out in order to reduce computation time. The merge operation implemented is the "modified merge" operation found in Brumbaugh-Smith *et al.* [10]. This operation is in linear time as a function of the sizes of the two sets to be merged.

2.4 The improvements

We originally had two suggestions for improvements, referred to as Condition I and II, that were both based on the idea of omitting "expensive" edges before the **Merge** in the algorithm. At each iteration in the routine, we are looking at an edge (i,j) from some node i to another node j, see Figure 4.

2.4.1 Condition I

The Condition I is a fast predomination check, which rules out "expensive" edges by considering the present set of labels. Consider again two particular nodes, i and j, and

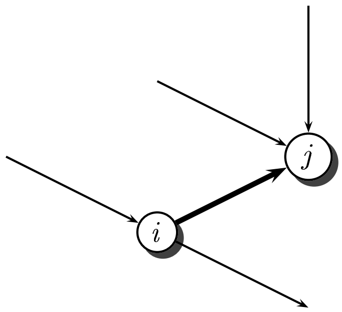

Figure 4: Evaluating the (i, j)-edge.

the sets of labels $D(i)$ and $D(j)$ at these two nodes. Assume that the two label-sets are non-empty, and that

$$D(i) = \{(c_1(i), t_1(i)), \cdots, (c_k(i), t_k(i))\} \quad \text{and} \quad D(j) = \{(c_1(j), t_1(j)), \cdots, (c_q(j), t_q(j))\}$$

with

$$\begin{array}{ccc} c_1(i) < c_2(i) < \cdots < c_k(i) & \text{and} & t_1(i) > t_2(i) > \cdots > t_k(i) \\ c_1(j) < c_2(j) < \cdots < c_q(j) & \text{and} & t_1(j) > t_2(j) > \cdots > t_q(j) \end{array}$$

We are now looking at the edge from node i to node j. Consider the two distinct but similar situations:

- Assume that $c_1(i) + c_{ij} \geq c_q(j)$. In this case we have:

$$\begin{array}{rcl} c_1(j) < c_2(j) < \cdots < c_q(j) & \leq & c_1(i) + c_{ij} < \cdots < c_k(i) + c_{ij} \\ t_1(j) > t_2(j) > \cdots > t_q(j) & ? & t_1(i) + t_{ij} > \cdots > t_k(i) + t_{ij} \end{array}$$

  So, if $t_k(i) + t_{ij} \geq t_q(j)$, then the set $D(i) + (c_{ij}, t_{ij})$ is dominated by the set $D(j)$. In fact, the set $D(i) + (c_{ij}, t_{ij})$ is dominated by the last label q of $D(j)$. As a merge of the two sets will return the set $D(j)$ unchanged, we can discard the edge between i and j, and proceed to the next edge.

- Assume that $t_k(i) + t_{ij} \geq t_1(j)$. In this case we have:

$$\begin{array}{rcl} c_1(i) + c_{ij} < \cdots < c_k(i) + c_{ij} & ? & c_1(j) < c_2(j) < \cdots < c_q(j) \\ t_1(i) + t_{ij} > \cdots > t_k(i) + t_{ij} & \geq & t_1(j) > t_2(j) > \cdots > t_q(j) \end{array}$$

  So, if $c_1(i) + c_{ij} \geq c_1(j)$, then the set $D(i) + (c_{ij}, t_{ij})$ is dominated by the set $D(j)$, because it is dominated by the first label of $D(j)$.

This simple condition which twice compares two numbers that have already been calculated can save a lot of cpu-time. How much time that is saved depends on the network structure. Section 2.5 summarizes this discussion from Paper B. Paper B also includes a discussion of how to generate random networks.

2.4.2 Condition II

The Condition II is inspired by an article by Tung and Chew [72]. The idea is to initialize node information from the terminal node in order to find the cheapest and fastest paths from an intermediate node j to the terminal node t, for all $n-1$ intermediate nodes. This initialization finds some upper bounds on the two objectives at the nodes, namely $(c^*, \hat{t})(j)$ for the cheapest path and $(\hat{c}, t^*)(j)$ for the fastest path. $c^*(j)$ is the cost of the cheapest (j,t)-path and $\hat{t}(j)$ is the corresponding upper bound on the time. Notice that the upper bounds on the (s,t)-path becomes $(\hat{c}, \hat{t})(s)$. The idea is illustrated in Figure 5.

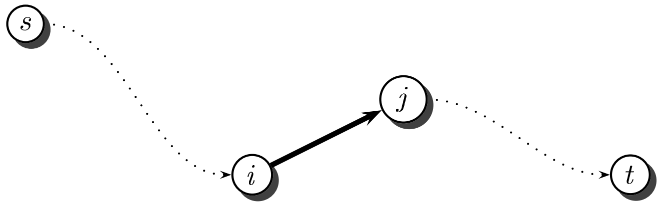

Figure 5: Illustrating the idea of Condition II.

If the present cheapest label at node i, plus the cost of the (i,j)-edge, plus the least cost for the remainder of the (j,t)-path, exceeds the upper bounds on the cost, the edge (i,j) can be left out of further consideration. Similarly with respect to time. Mathematically we get the following two conditions to discard an edge:

$$c_1(i) + c_{ij} + c^*(j) > \hat{c}(s) \text{ or } t_k(i) + t_{ij} + t^*(j) > \hat{t}(s) \tag{5}$$

Unfortunately the initialization of the node information turned out to be too time-consuming, and the bounds were to loose because they are upper bounds on all $s-t$ paths and not subpaths as in Condition I. The initialization is done by running Dijkstra's shortest path algorithm twice, with the edge orientation in the opposite direction. If Dijkstra's algorithm is too slow, this indicates that the Label Correcting algorithm is indeed fast. The bounds being too loose means that there were almost never any edges discarded by (5).

2.5 Computational results

We have tested the improved algorithm (*alg2*) together with the Brumbaugh-Smith algorithm (*brum*) of Section 2.3. All algorithms are implemented in C++, and can be found

on the homepage http://home.imf.au.dk/ajs/. We have used an HP 9000 series computer with a single processor. For each size of network we have used 10 random networks, generated with NETMAKER (see Paper B).
The objective is to evaluate the effectiveness of Condition I on networks with different characteristics (density). The density of a network is the relationship between the number of nodes and the number of edges. If parallel edges are not allowed, the number of edges in a connected network is between $n-1$ (tree) and $n(n-1)/2$ (complete).

# nodes	*brum*	Merges	*alg2*	Condition I's	% Merges in *alg2*	% cpu-time
200	9.01	761.30	4.12	208.40	46.49	45.76
400	40.38	1615.20	20.96	407.20	50.09	51.91
600	92.96	2502.00	51.40	578.15	52.95	55.29
800	187.05	3385.00	111.82	757.65	54.01	59.78
1000	280.61	4668.20	162.52	970.90	57.80	57.92

Table 3: Cpu-times, number of Merges and number of Condition I's for *brum* and *alg2* when the number of outgoing edges are between 1 and 3 at each node.

The first comparison is made on a sparse network, where the average number of edges is only two times the amount of nodes. The results are shown in Table 3, and the overall conclusion is that *alg2* is considerably faster than *brum*.
There is one implementation detail that is important to mention. The *brum* algorithm is implemented directly as it is described in Brumbaugh-Smith *et al.* [10]. In *alg2* a node with an empty label-set automatically gets the label-set plus the edge-weights from the first predecessor node. When the algorithms are compared, the number of nodes (minus one) is added to the number of Condition I's, because this is the number of Merge operations saved. So in Table 3 with 400 nodes, the "% Merges in *alg2*" is calculated as

$$\left(1 - \frac{407.20 + 399}{1615.20}\right) * 100 = 50.09$$

It can also be seen that, as the number of nodes increases, the fraction of Condition I's decreases. This is due to the fact that the probability of Condition I being fulfilled decreases as the label-sets increase. The label-sets increase in size as we move towards the terminal node, and in the larger networks, the average number of nondominated values is higher and therefore the label-sets are bigger. As expected *alg2* performs very well on sparse networks, because of the small label-sets.

Next we look at less sparse networks with an average number of 3 outgoing edges per node. The results are shown in Table 4, and as expected the fraction of Condition I's has

# nodes	*brum*	Merges	*alg2*	Condition I's	% Merges in *alg2*	% cpu-time
200	18.34	1399.65	12.16	303.25	64.12	66.29
300	45.37	2221.4	31.76	428.45	67.25	70.00
400	80.43	3080.7	58.14	487.35	71.23	72.28
500	129.77	4006.65	96.91	652.10	71.27	74.68
800	336.65	6801.80	245.77	933.95	74.52	73.00

Table 4: Cpu-times, number of Merges and number of Condition I's for *brum* and *alg2* when the number of outgoing edges are between 2 and 4 at each node.

dropped. Because the cpu-time saved is fairly proportional to the number of Condition I's, *alg2* only performs about 25-35 % better than the *brum* algorithm for networks with this density (and this size).

# nodes	*brum*	Merges	*alg2*	Condition I's	% Merges in *alg2*	% cpu-time
100	12.59	2796.1	11.05	251	87.48	87.76
200	79.55	6055.40	73.50	284.4	92.02	92.40
300	195.48	9680.60	183.55	346.45	93.33	93.90
400	349.04	13733.30	329.83	430.25	93.96	94.50
500	589.84	17943.05	558.87	463.40	94.64	94.75

Table 5: Cpu-times, number of Merges and number of Condition I's for *brum* and *alg2* when the number of outgoing edges are between 7 and 15 at each node.

For the dense networks of Table 5 with an average of 11 outgoing edges per node, the fraction of Condition I's is much smaller. The cpu-times are again proportionately faster as well. This table illustrates that even in dense networks there are still cpu-time saved by imposing the condition. We therefore conclude that the cost in cpu-time of checking the condition is negligible.

2.6 Conclusions on the BSP problem

It seems that the Label Correcting approach is the best for the BSP problem. Even though the problem is $\mathcal{NP}$-complete the solution methods are usually quite fast. As noted, a separate first phase to find the supported solutions using the weighting method does not seem worthwhile. However, even though the Label Correcting method is fast, there is still space to speed up the algorithm, i.e. Condition I.
Another positive feature about the Label Correcting method, is that it easily generalizes to more than two objectives. All that needs to be modified is the Merge operation. Unfortunately Condition I does not generalize to more than two objectives.

3 Bicriteria combinatorial Max-Ordering (MO) problems

Max-ordering (MO) problems are multicriteria optimization problems in which the goal is to minimize the worst of several objective functions. They can be formulated as follows.

$$\min_{x \in S} \max_{i=1,\ldots,Q} f^i(x), \tag{6}$$

where $f^i(x)$ denotes the objective functions of the problem. The problem is denoted max-ordering instead of min-max in order not to confuse terminology with single objective problems, i.e. $\min_{x \in S} \max_{e \in x} w_e$ which finds solutions where the largest weight is minimal, e.g. the path where the largest edge-weight is minimal. Max-ordering problems arise in various applications, see Rana and Vickson [62] or Warburton [76], and as subproblems in interactive methods for the solution of multicriteria optimization problems such as the GUESS method (Buchanan [11]), STEM (Benayoun *et al.* [5]), and the interactive weighted Tchebycheff method (Steuer and Choo [69]).

In this paper we consider max-ordering problems in a combinatorial context, i.e. we assume that S is a finite set, e.g. the set of paths between two nodes of a network or the set of spanning trees of a graph.

There is a number of previous research papers on this topic (Ehrgott [23], Hamacher and Ruhe [37], Murthy and Her [56], Ehrgott *et al.* [26]) and see Ehrgott and Gandibleux [25] for more. Various authors observed that, even in the bicriteria case, max-ordering problems are usually $\mathcal{NP}$-complete. The methods proposed for their solution include branch and bound (Rana and Vickson [62]), labeling algorithms (for shortest path problems) (Murthy and Her [56]) and ranking methods (Ehrgott [23], Hamacher and Ruhe [37]) - that is the application of algorithms to find K best solutions of (single objective) combinatorial problems.

We also propose methods involving ranking algorithms actually overcoming the main problem of the method proposed in Hamacher and Ruhe [37], at least for the case of two objectives, see Remark 1. We combine the ranking method with the two-phases method originally developed for the determination of all efficient solutions of bicriteria combinatorial optimization problems, Ulungu and Teghem [74].

In Sections 3.1 and 3.2 we study the biobjective case, $Q = 2$, and in Section 3.4 we present a heuristic for Phase 1 in the multiobjective case, $Q > 2$.

3.1 Theoretical motivation

We shall use the notation $g(x) = \max\{f^1(x), f^2(x)\}$ for the max-ordering objective value of a feasible solution $x \in S$. Next we present three basic results. The first one is well-known, see e.g. Hamacher and Ruhe [37].

Lemma 1 *There is at least one optimal solution of the max-ordering problem* $\min_{x \in S} g(x)$ *which is efficient.*

The next Lemma is specifically stated for two objectives. It formalizes the argument that the maximum of two functions is minimal, if the objective values are as equal as possible. Its proof is immediate from the definition of the max-ordering problem.

Lemma 2 *Let* $\mathcal{X}_{Par} = \{x^1, \ldots, x^p\}$ *be the set of efficient solutions of a bicriteria combinatorial optimization problem. Assume that* $f^1(x^i) \leq f^1(x^{i+1})$ *and* $f^2(x^i) \geq f^2(x^{i+1})$ *for* $1 = 1, \ldots, p-1$ *and define* $K := \min\{i : f^2(x^i) < f^1(x^i)\}$. *Then the following hold.*

1. *If* $K = 1$, x^1 *solves the max-ordering problem.*
2. *If* $K = \infty$, x^p *solves the max-ordering problem.*
3. *Otherwise* x^K *or* x^{K-1} *(or both) solve the max-ordering problem.*

A special case occurs if there is an efficient solution with both objectives equal.

Lemma 3 *If there is an efficient solution* x *such that* $f^1(x) = f^2(x)$, *then* x *also minimizes* $g(x)$.

These three lemmas state that we can restrict our search for a solution for a minimizer of $g(x)$ to efficient solutions, with their two objectives as equal as possible. In other words, efficient max-ordering solutions will be located close to the halving line $f^1 = f^2$ in criterion space.

3.2 Two-phases algorithm

First, we look for the two supported efficient solutions for which $f^1(x^i) \leq f^2(x^i)$ and $f^1(x^{i+1}) > f^2(x^{i+1})$ according to the order of Lemma 2. To do so, we start with solutions x^1 and x^2 minimizing objectives f^1 and f^2, respectively. We then proceed to solutions where the difference of objective values is smaller. When this is no longer possible, we will either have one supported efficient solution with $f^1(x) = f^2(x)$, or we end up with two neighboring supported efficient solutions, say x^1 and x^2 such that $f^1(x^1) < f^2(x^1)$ and $f^1(x^2) > f^2(x^2)$. According to Lemma 3, the first case solves $\min_{x \in S} g(x)$, and any other efficient solution must have one objective value smaller and one bigger than $g(x)$. Of course, it may happen that one of the objectives dominates the other completely, i.e.

$\min_{x \in S} f^1(x) \geq \max_{x \in \mathcal{X}_{Par}} f^2(x)$ (cases 1 or 2 in Lemma 2). In this case the problem is trivial, and we can easily detect it when computing x^1 and x^2 for the first time.
Should we terminate Phase 1 with two solutions, we will have to investigate unsupported solutions in the right-angled triangle defined by the hyperplane through the point $f(x_{cur})$ with normal λ and $(g(x_{cur}), g(x_{cur}))$, where x_{cur} is the current best solution, see Figure 7. For this we use the ranking algorithm. In fact, $f(x^1)$ and $f(x^2)$ uniquely define weights λ_1, λ_2 such that both $x^1$and x^2 are optimal solutions of

$$\min_{x \in S} \lambda_1 f^1(x) + \lambda_2 f^2(x).$$

We can now apply a ranking algorithm to find second, third, ... best solutions for this problem, in order to find unsupported solutions in the identified triangle. A similar procedure was proposed for the identification of all unsupported efficient solutions in Coutinho-Rodrigues *et al.* [19].
The algorithm will stop if we encounter a solution x with $f^1(x) = f^2(x)$, as this must be the optimal solution we are looking for, or $\lambda_1 f^1(x) + \lambda_2 f^2(x) \geq g(x_{cur})$, since no further solution will be in the triangle and therefore no longer a candidate for a MO optimal solution. In the latter case, the currently best solution is the optimal solution of the max-ordering problem.

The idea of the first phase is illustrated in Figure 6. With solutions x^1 and x^2 we compute the normal to the line connecting $f(x^1)$ and $f(x^2)$. This normal serves as a weighting vector for combining the two objectives, and its negative is the direction in which we search for a new supported efficient solution which is eventually found at x^3 with objective values $f(x^3)$.

Remark 1 *The values λ_1, λ_2, identified at the end of Phase 1, are the best choice of λ in the method proposed by Hamacher and Ruhe [37] and will overcome the problem that for an unfortunate choice of λ, that method turns out to be a complete enumeration of all feasible solutions.*

We illustrate the algorithm on an example. In Figure 7 we show the objective values of 6 feasible points indexed in the order of their generation.
In Phase 1, x^1 and x^2 will be generated first. Weights λ_1 and λ_2 are computed corresponding to the normal to a line connecting $f(x^1)$ and $f(x^2)$ and $x_{cur} = x^2$. Solution of the weighted sum problem results in x^3. Since $f^1(x^3) < f^2(x^3)$, x^1 is replaced by x^3. The current best x_{cur} is updated to x^3. The second weighted sum problem uses updated λ's corresponding to the normal of the line connecting $f(x^2)$ and $f(x^3)$. Assume x^3 is returned

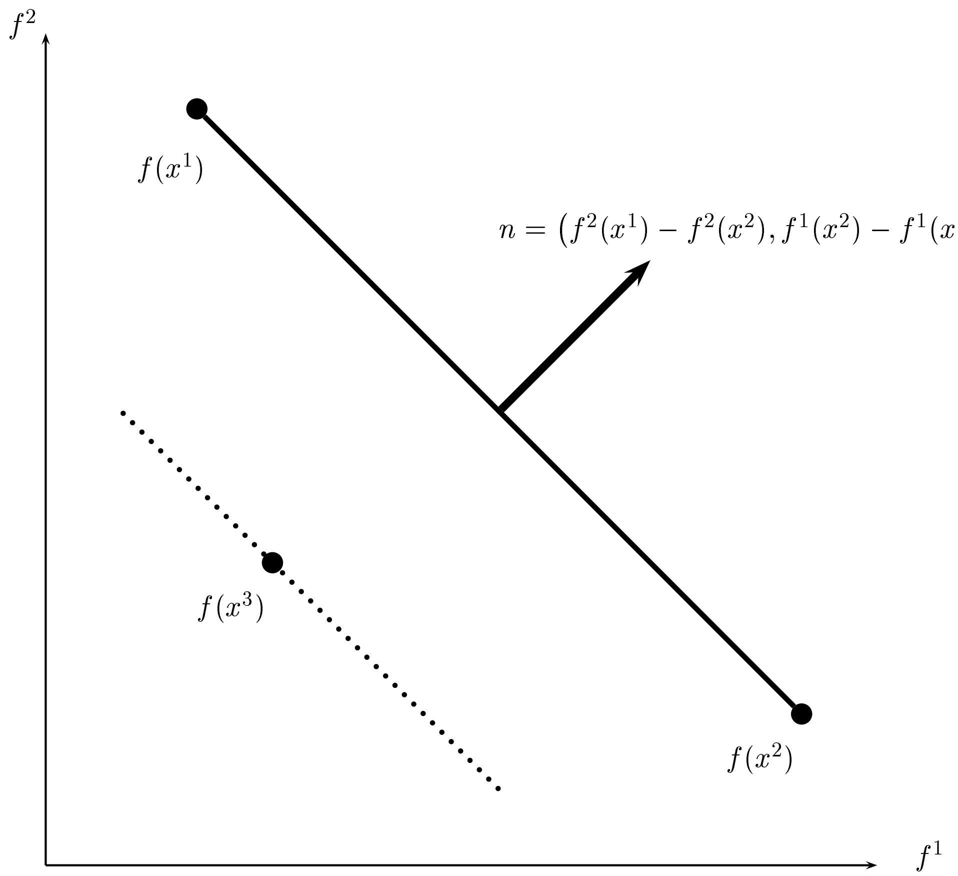

Figure 6: Illustration of search direction in Phase 1

as an optimal solution. Thus no new supported efficient solution is found, and we continue with Phase 2 to investigate the earlier defined triangle. Note that the supported solution x^4 is not generated in Phase 1.

We know that x^3 and x^2 are first and second best solutions of the weighted sum problem, therefore we are searching for the third best solution by searching in direction λ. This turns out to be x^4, which is discarded as not being in the triangle $\big(f^2(x^4) > f^2(x^3) = g(x_{cur})\big)$. So we set $K = 4$, identify x^5 as the next solution, and this passes all tests. In our example x^5 replaces x^3 as the current best solution and K is set to 5. The next solution is x^6, the combined objective value of which is larger than that of the third corner point of the triangle. We will therefore find no further points in the triangle and stop with the optimal solution $x^* = x_{cur} = x^5$.

Remark 2 *In Phase 2 the following situation may occur: The solution of the weighted sum problem is another supported efficient solution which is, as x^1 and x^2, optimal for the weighted sum problem. Its objective function vector lies on the line between $f(x^1)$ and*

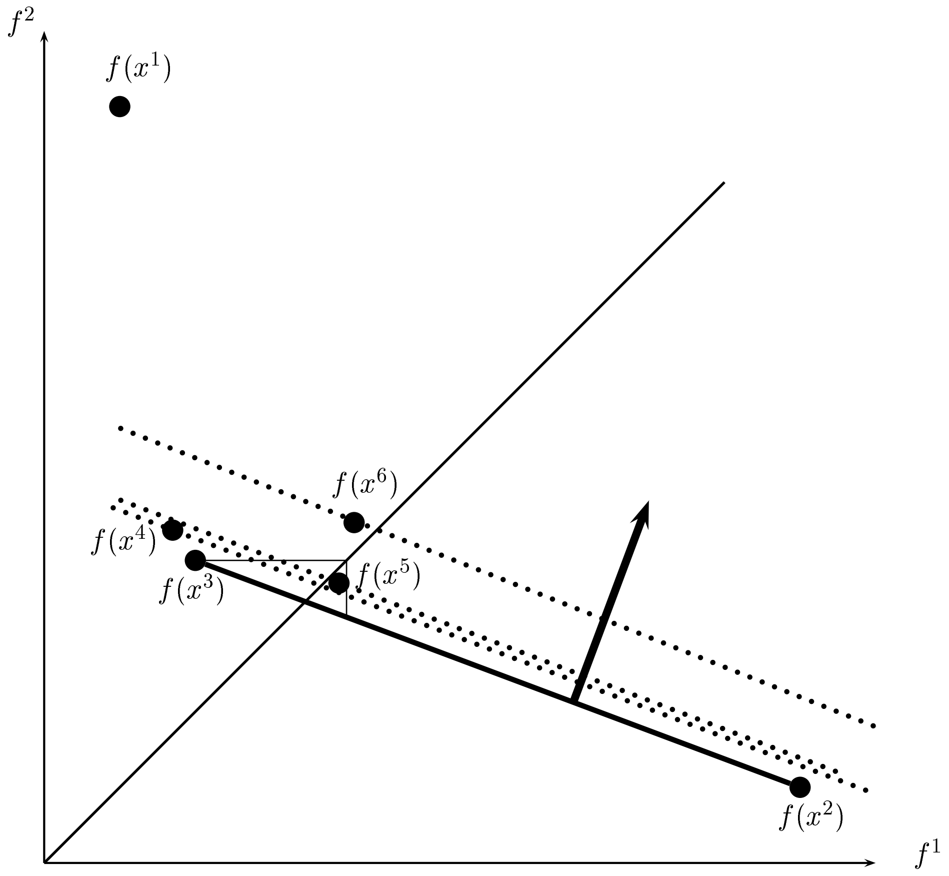

Figure 7: Illustrative example

$f(x^2)$. In this case, this point creates two new and smaller triangles. We can restrict search to the one which is intersected by the halving line $f^1 = f^2$.

3.3 K-best algorithms

As we propose the use of ranking algorithms, our method is obviously restricted to such combinatorial optimization problems for which efficient methods for finding K-best solutions are available. We briefly review some of these here.

The largest amount of research on ranking solutions is available for the shortest path problem. Algorithms developed by Azevedo et al. [3], Martins *et al.* [52] or Eppstein [27] are very efficient. The best complexity known is $O(m+n\log n+K)$ by Eppstein's method. However, numerical experiments reported by Martins *et al.* [51] show their algorithm to be very competitive. Its complexity is $O(m + Kn\log n)$.

The second problem for which several methods are known, is the minimum spanning tree problem. We mention papers by Gabow [32] and Katoh *et al.* [46]. The best known complexity is $O(Km + \min(n^2, m \log\log n))$.

In the seventies and eighties some general schemes for ranking solutions of combinatorial optimization problems have been developed by Lawler [48] and Hamacher and Queyranne [39]. The application of the latter led to algorithms for matroids (Hamacher and Queyranne [39]), with the special case of uniform matroids discussed in Ehrgott [23]. The complexity of the latter is $O(K(n+m) + \min\{n \log n, nm\})$. Finally, an algorithm to rank (integer) network flows was presented in Hamacher [35]. Its complexity is $O(Knm^2)$. We note that only algorithms allowing the construction of solutions with the same objective function values are applicable in our method. This is evident from the fact that at the beginning of Phase 2, we have x^1 and x^2 as optimal, i.e. first and second best solutions of the weighted sums problem.

3.4 Phase 1 heuristic for the multiobjective case

A natural question is the extension of the algorithm to more than two objectives. With such an endeavor we encounter two major difficulties. The first one being that problems with at least three objectives cannot be reduced to subproblems with two objectives only. Thus, in the multicriteria case all criteria have to be considered simultaneously.

Example 1 *Consider a combinatorial problem with three objectives and the following set of nondominated vectors*

$$\left\{ \begin{pmatrix} 7 \\ 5 \\ 3 \end{pmatrix}, \begin{pmatrix} 6 \\ 4 \\ 8 \end{pmatrix}, \begin{pmatrix} 9 \\ 4 \\ 2 \end{pmatrix}, \begin{pmatrix} 6 \\ 8 \\ 2 \end{pmatrix} \right\}$$

The unique max-ordering solution is the first one, with $g(x) = 7$. However, looking at only two of the objectives at a time, we obtain the following. For f^1, f^2 only, the minimal value of $g(x)$ is attained at the second solution, for f^2, f^3 it is the third, and for f^1, f^3 it is the fourth. Thus none of the bicriteria subproblems yield the true MO optimal solution.

The second major difficulty is in the generalization of Phase 1. This problem has been observed by many researchers applying the method for the generation of all efficient solutions. In contrast to the bicriteria case, there may exist supported nondominated points, which lie above (rather than below) a previously constructed hyperplane. For a discussion see Solanki *et al.* [67].

Therefore, we present a heuristic for Phase 1, when we have more than two objectives. Since it is not known how to determine appropriate λ-weights for the weighted objective function

$$W(x, \lambda) = \sum_{i=1}^{Q} \lambda_i f^i(x)$$

in order to find all the supported solutions, we have to settle for a heuristic procedure to produce a good λ. We are still guaranteed to find the MO optimal solution, since the K best procedure in Phase 2 can go through all solutions, independent of our choice of λ.
The idea of the algorithm is also to search for an MO optimal solution. Remember that a good current best solution x_{cur}, limits the search in Phase 2. During Phase 1 we modify an initial λ to search in the direction where the current best solution is found. This is done by increasing the λ_j-weight for the objective where the current maximum is attained, and decrease the λ_i-weight for the objectives that are currently small enough. This leads to the following relation between the λ's in iteration p and $p+1$ if $arg \max_i f^i(x^p) = j$:

$$\lambda_j^{p+1} > \lambda_j^p \text{ and } \lambda_i^{p+1} \leq \lambda_i^p \ \forall i \neq j$$

In the following two sections we discuss some possible initialization and stopping criteria for the Phase 1 heuristic. Pseudo code for the complete Phase 1 is presented in Section 3.4.3.

3.4.1 Initialization

First we solve the Q single objective problems, to see if any single objective dominates, and construct the $Q \times Q$ pay-off table. Let $\epsilon > 0$ be a very small positive constant, which is used to avoid weakly efficient solutions.

1. For $i = 1$ to Q do

$$\min_{x \in S} f^i(x) + \epsilon \sum_{j \neq i} f^j(x), \text{ with optimum in } x^i$$

The set of $x^1, \ldots, x^Q$ solutions corresponding to the above Q optimal solutions is denoted X_{pay}. If $\exists\ i \in Q$ such that

$$f^i(x^i) \geq f^j(x^i) \ \ \forall j \neq i$$

then stop with $x^* = x^i$ as optimal (trivial) solution, and $g^* = f^i(x^*)$. In the rest of this section assume this is not the case, i.e. $\forall i \in Q\ \exists j \neq i$ such that $f^i(x^i) < f^j(x^i)$.
It may be useful to have a lower and an upper bound on g. Therefore we define such bounds during the initialization.

$$g_{LB} = \max_i f^i(x^i) \quad \text{and} \quad g_{UB} = \min_{x \in X_{pay}} \max_i f^i(x) = \min_{x \in X_{pay}} g(x)$$

The lower bound is where we have the minimal largest value of the extreme solutions, and the upper bound is where the initial largest value is as small as possible. The x corresponding to g_{UB} is the current best solution x_{cur}. Notice that g_{UB} is an upper bound on **all** the individual objectives with respect to the MO problem. The bounds may be used as a Phase 1 stopping criterion, see Section 3.4.2.

Finally, we need to find an initial λ. We can think of λ_i as the weight of the i'th objective. Our goal of Phase 1 is a fixed best λ. We initialize bounds for λ_i $\forall i$ as

$$\lambda_{iLB} = 0 \text{ and } \lambda_{iUB} = 1 \ \ \forall i$$

These bounds on the λ-weights will be modified during the algorithm, see Section 3.4.3. Now we define the starting λ^1. We have two suggestions:

- $\lambda^1 = (\frac{1}{Q}, \ldots, \frac{1}{Q})$.
- $\delta^i = \bar{f}^i \ \forall i$, where $\bar{f}^i = \max_{x \in X_{pay}} f^i(x)$. Let $\delta = \sum_i \delta^i$. Now we define λ^1 as $\lambda^1 = (\frac{\delta^1}{\delta}, \ldots, \frac{\delta^Q}{\delta})$. In this way the numerically large objectives get more attention initially.

To summarize the initialization, we present it in four steps:

1. Calculate pay-off table.
2. Check for trivial solution.
3. Find initial values for x_{cur}, g_{LB} and g_{UB}.
4. Find initial λ^1.

3.4.2 Stopping criterion

Due to the difficulties of determining the supported solutions in Phase 1, a stopping criterion is not straightforward in the multiobjective case. We discuss some alternatives in this section, and it may be possible to have more than one stopping criterion in the algorithm. Let us list some possibilities.

- If the same solution is repeated a certain number of times, $x^p = x^{p+1} = \ldots = x^{p+t}$ for some predetermined t.
- If the maximum deviation on the λ bounds is sufficiently small, $\max_i \lambda_{iUB} - \lambda_{iLB} < \epsilon$, for some predefined tolerance level ϵ.
- If the gap between the upper and lower bound is sufficiently small, $g_{UB} - g_{LB} < \delta$, for some predefined tolerance level δ.

- Run Phase 1 for at most T iterations.

There may, of course, be other alternatives as well.

3.4.3 The algorithm

1. Initialization, see Section 3.4.1.
2. Set $p = 1$ (iteration counter).
3. Solve $\min_{x \in S} \sum_i \lambda_i^p f^i(x)$ and let x^p be the optimal solution.
4. If $g(x^p) < g(x_{cur})$ then $x_{cur} = x^p$ and $g_{UB} = g(x^p)$.
5. Check stopping criterion, see Section 3.4.2.
6. Assume $arg \max\limits_i f^i(x^p) = j$, i.e. the j'th objective needs more attention.
7. $\lambda_{jLB} = \lambda_j^p$ and $\lambda_{iUB} = \lambda_i^p \; \forall i \neq j$.
8. Define $\Delta_i = \lambda_{iUB} - \lambda_{iLB} \; \forall i$ and $\Delta = \sum\limits_{i \neq j} \Delta_i$.
9. $\lambda_j^{p+1} = \lambda_{jLB} + \frac{1}{2}\Delta_j$ and $\lambda_i^{p+1} = \lambda_{iUB} - \frac{1}{2}\Delta_j \frac{\Delta_i}{\Delta} \; \forall i \neq j$.
10. p=p+1 go to 3

This concludes Phase 1. The steps 6 - 9 modify the λ-bounds to decrease the interval of each possible λ_i value.
In Phase 2 we use the K best procedure with the appropriate λ^p from the end of phase one. The stopping criterion in phase two is the same as in the biobjective case, namely $\sum\limits_i \lambda_i^p f^i(x^K) \geq g(x_{cur})$.

3.5 Conclusions on MO problems

At first the max-ordering objective may not seem interesting, but it appears as a subproblem in several well-known MCDM methods. Therefore, this problem is worth studying more carefully. We have found a procedure for the solution of the max-ordering problem for combinatorial problems. The effectiveness of the procedure depends on the particular problem. Phase 1 works well if the single objective combinatorial problem is easy, such as the shortest path problem. Phase 2 works well if the K best procedure works well.
In the case with more than two objectives, the Phase 1 procedure is without guarantees of finding the best supported efficient solutions. However, the K best procedure will (eventually) find the MO optimal solution, even with a poor choice of λ.

4 Approximate solution of semi-obnoxious location problems

In two of the traditional single facility location problems, a new facility is located (placed) so as to minimize transportation costs (minisum), or as to minimize the distance to the farthest customer (minimax). In the minisum problem we sum all the distances between the new facility and the customers, multiplied by a weight depending on the individual customer. In the minimax problem we minimize the largest weighted distance. A traditional example of the minisum model is the location of a warehouse and an example of the minimax model is locating a fire station. These models are presented in Love *et al.* [49] and Francis *et al.* [31], both including many references. The obnoxious location problem is a more recent class of problems, where the two most common ones are the maxisum and maximin models. When locating an obnoxious (undesirable) facility, the goal is to place it as far from the existing facilities (demand points, customers) as possible. See Erkut and Neuman [28] or Carrizosa and Plastria [14] for a review.

Instead of classifying the problem as obnoxious or desirable, the models can be divided into planar and network models by their structure, and not their objectives. This partition makes it more easy to list existing literature. Some references on planar models are [30], [37], [38], [59], [60], [61] and [64], and some references on network models are [6], [7], [15], [21], [22], [36], [41], [42], [47], [54], [57] and [58].

There is little literature combining the desirable and the obnoxious facility location models, even though many facilities are both obnoxious and desirable. An airport is obviously desirable for the travelers, but obnoxious for the nearby citizens. In this section we model the combined problem as a Bicriterion Semi-obnoxious Location (BSL) problem. One objective function is obnoxious and one is desirable. We consider both the planar case (Section 4.1) and the network case (Section 4.2) of the problem. In the network case where the demand points are nodes in a network and we try to locate the new facility in a node or on an edge, we have found no references to earlier work. However, new results are presented in Section 5 (and Paper D). In the planar case, where the feasible locations are in R^2, we have found only three references, namely two papers by Brimberg and Juel, [8] and [9], and one paper by Carrizosa *et al.* [13].

The theory of the planar and network models is quite different, and the two models are not often compared, even though they often try to describe the same real-life problem. We briefly compare the two models in Section 4.3.

4.1 The planar case : The BSPL problem

We formulate the Bicriterion Semi-obnoxious Planar Location (BSPL) problem in the following way. There are n facilities (demand points) located at points $a_1, a_2, \dots a_n$, and the objective is to locate a semi-obnoxious facility at x so as to minimize a weighted sum of the distances raised to a negative power, and to minimize the weighted sum of the distances between the existing facilities and the new facility. The first criterion may be thought of as a pollution effect and the second criterion as transportation costs.

$$\begin{array}{ll} \min & f(x) = \sum\limits_j w_j^1 (\| x - a_j \|_{p_1})^{-b}, \qquad b > 0 \\ \min & g(x) = \sum\limits_j w_j^2 \| x - a_j \|_{p_2} \\ \text{s.t.} & \\ & x \in S \end{array} \tag{7}$$

where $\| x - a_j \|_p = (|x_1 - a_{j1}|^p + |x_2 - a_{j2}|^p)^{1/p}$ is the usual l^p norm, $p \geq 1$.

We prefer this obnoxious function, because it minimizes the overall obnoxiousness when far from a demand-point, but reflects the local effects when close to a demand-point. Corresponding to this objective we use the weights w^1. The second objective is the standard formulation for locating an attractive facility by minimizing the weighted sum of the distances (called minisum or median). Please note that we use weights w^2 with this objective, so that the two objectives may be weighted differently with respect to each of the n demand points. We assume that all weights are non-negative.

If we are modeling where to place a new airport (example in Paper C), the first weight w_j^1 may depend on the population at demand point j (e.g. city), and the second weight w_j^2 may be the expected number of passengers on a yearly basis from demand point j.

S is the set of feasible solutions. Because of the obnoxious effects from the new semi-obnoxious facility, we assume that it is forbidden to place it too near an existing facility. Therefore, we require, that $\| x - a_j \|_{p_1} > \epsilon, \quad j = 1, \dots, n$, where ϵ is a small positive number. Note that this assumption makes the two objective functions Lipschitzian in the feasible set S.

Since the obnoxious objective function $f(x)$, is a slightly complicated function, we will settle for an approximation of the efficient set $\mathcal{X}_{Par}$. To obtain this approximation we will apply the BSSS method first introduced by Hansen *et al.* [44].

4.1.1 The idea of the Big Square Small Square (BSSS) algorithm

Since we apply the BSSS method to solve the BSPL problem (and also to the BSNL problem), we will outline the idea of the method.

Suppose that the feasible region S is contained in a disjoint union of squares of equal size. Each of these squares are considered separately. Consider one of the squares, say Q_i. We divide Q_i into four sub-squares Q_{i1}, Q_{i12}, Q_{i3} and Q_{i4} of equal size. For each of these sub-squares, say Q_{i1}, lower bounds on the objective function values $(f(x), g(x))$, $x \in Q_{i1}$, are found. By comparing this lower bound with a sample set of objective function values, it may be determined that square Q_{i1} contains only inefficient points. If this is the case, square Q_{i1} is called an inefficient square and may be deleted from further consideration. The squares that cannot be classified as inefficient are put into the list and will later be divided further into four new sub-squares. The process continues until the side-lengths of all the remaining squares (those that are not classified as inefficient) in the list are below a pre-specified value ϵ. The idea is illustrated in Figure 8 below. The output from the algorithm is an ordered set of "efficient" squares.

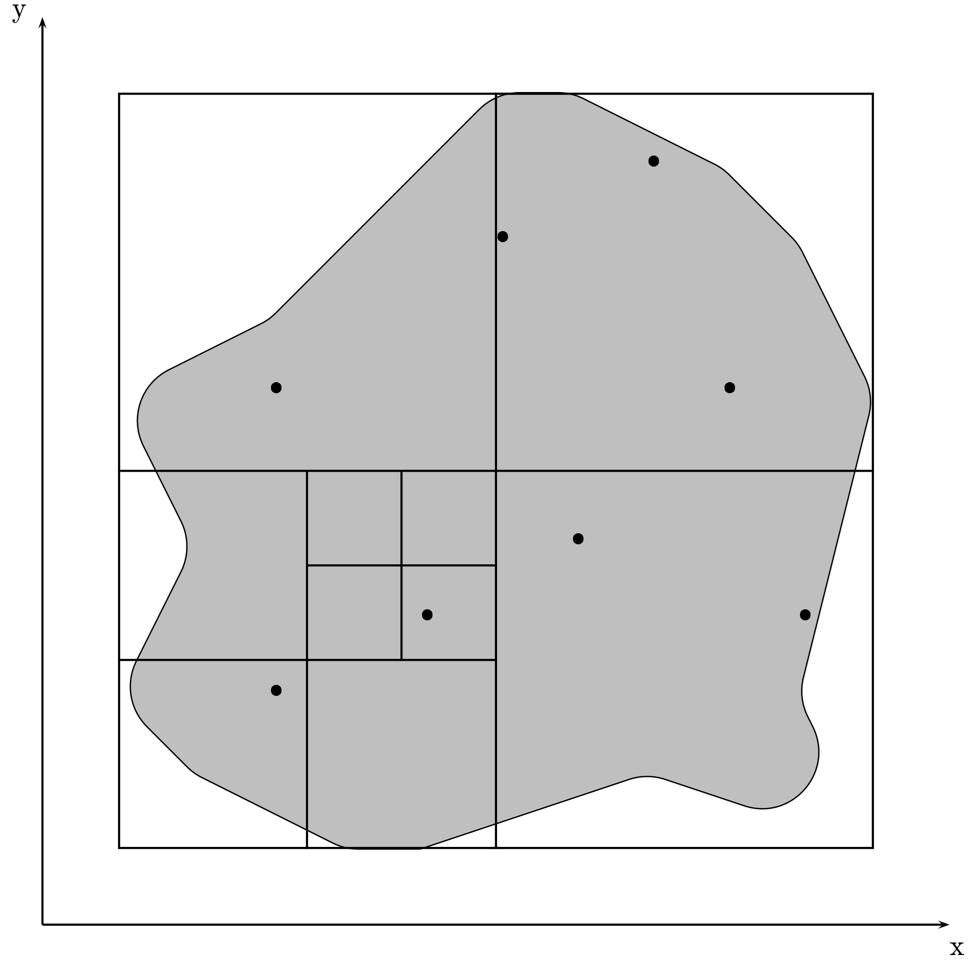

Figure 8: BSSS idea

A few comments on the procedure are appropriate. The sample list of objective function values are used to dominate sub-squares with poor objective function value bounds. Therefore, the values should in a way represent the objectives' behavior over the feasible region. This is done by calculating objective function values in the centers of all the squares, and then deleting pairs of objective function values being dominated by other objective function values. If the center of a particular square is not in S, we simply ommit this calculation.

It is also essential that we use good lower bounds for the objective function values over the squares. If the bounds are poor, the convergence of the algorithm may be slow, because we will end up with a large number of squares. Fortunately, good bounds exist. These bounds are explained in detail in Sections 4.1.2 and 4.1.3.

Finally, we need to check if a square is contained in the feasible region, is overlapping the region or is outside the region. For a discussion of this issue we refer to the paper by Hansen *et al.* [43].

4.1.2 Calculating lower bounds

In order to calculate lower bounds on the two objectives, we use an approximation of the weighted distances. This distance approximation is illustrated in Figure 9 for the l^2 norm. The lower bound for the distance is found in Hansen *et al.* [43], and the upper bound for the distance is found in Hansen *et al.* [44].

The plane is divided into 9 regions, obtained by extending the four sides of Q_i. The regions are the square Q_i, the four side regions, and the four corner regions. The square Q_i will be in the center.

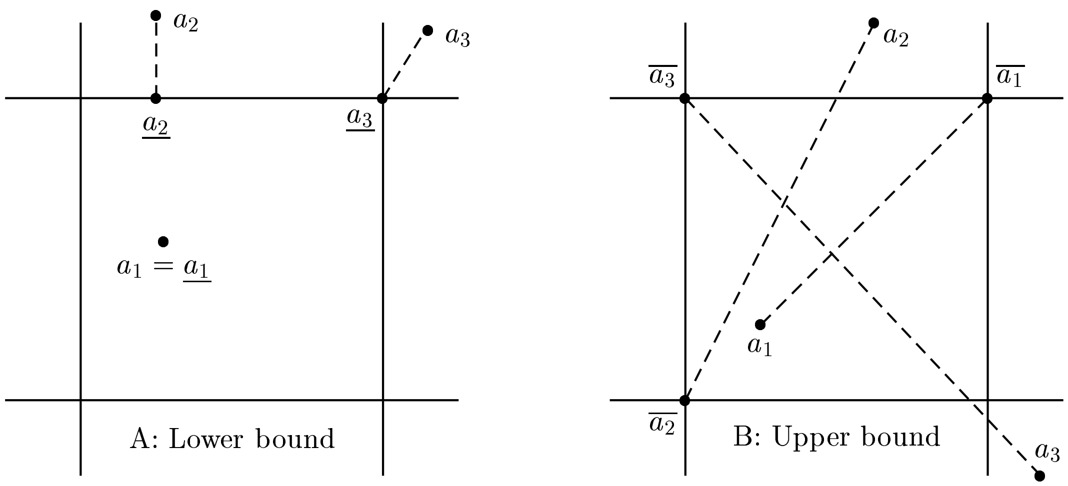

Figure 9: Lower and upper bounds on the distances.

Now let a_j be a particular location. With this location we associate a closest point $\underline{a_j} \in Q_i$ and a furthest point $\overline{a_j} \in Q_i$, see Figure 9. We may then calculate a lower bound on the values of f and g in Q_i as follows:

$$\begin{array}{rcll} \underline{f}(Q_i) & = & \sum_j w_j^1 (\| \overline{a_j} - a_j \|_{p_1})^{-b} & \text{Case B in Figure 9} \\ \underline{g}(Q_i) & = & \sum_j w_j^2 \| \underline{a_j} - a_j \|_{p_2} & \text{Case A in Figure 9} \end{array}$$

Clearly, $(\underline{f}(Q_i), \underline{g}(Q_i) \le (\min_{x \in Q_i} f(x), \min_{y \in Q_i} g(y))$. Therefore, we can use the bound $\underline{z}(Q_i) = (\underline{f}(Q_i), \underline{g}(Q_i))$ for efficiency checking in the algorithm. If at some point we have found a sample value $x \in S$, such that $(f(x), g(x)) < (\underline{f}(Q_i), \underline{g}(Q_i))$, then clearly all points in Q_i are dominated by x. It follows that square Q_i contains only inefficient points. Therefore it is not necessary to consider Q_i anymore. This bound approach can be used for any $p \in [1; \infty]$. Please note that the bounds obviously converge when the squares get smaller.

4.1.3 Exact lower bound

Since the minisum objective is a nice convex function, it is possible to calculate an exact lower bound for the squares in most situations. The level sets of a convex function are convex sets, and the gradient can therefore be used as follows.

For a square Q_i with corners c_1, c_2, c_3 and c_4, find the corner c_h with the minimum function value $g(c_h)$. If the direction of steepest descent "**points away**" from the square Q_i, then the lower bound $\underline{g}(Q_i)$ is exactly $g(c_h)$. By "pointing away" we mean that the direction of steepest descent has an angle of at least 90 degrees with the sides of Q_i, see case A in Figure 10. If this angle is less than 90 degrees then the minimum value over Q_i is not in c_h. Finally, if the direction points **into** Q_i, the minimum value is **not** in c_h but inside Q_i.

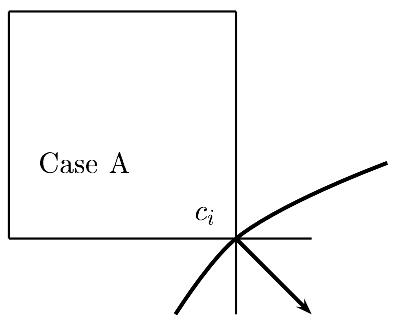

Figure 10: Exact lower bound, depending on directional derivative

From the above an exact lower bound can easily be computed, if the directional derivative points away from the square. We only need to compute four function values and the

directional derivative in the minimum value corner. The case A will occur in most of the evaluations, but not in all.
The directional derivative $g'(x_0, y)$ of g at $x_0 \in S$ in the direction y is defined as follows:

$$g'(x_0, y) = \nabla\, g(x_0) \cdot y$$

where $\nabla g(x_0)$ is the gradient of g evaluated in x_0.
If we consider the l^2 norm, the gradient looks as follows:

$$\nabla\, g(x_0) = \left(\sum_j \frac{w_j^2(x_{01} - a_{j1})}{\parallel x_0 - a_j \parallel}, \sum_j \frac{w_j^2(x_{02} - a_{j2})}{\parallel x_0 - a_j \parallel} \right)$$

Similar expressions can be found for the l^p norm, for $p \in [1; \infty]$. This reveals the well-known problem; if x_0 is at a demand point, the gradient is undefined because of the numerator being zero. This is not a problem in our case since $\parallel x - a_j \parallel_{p_1} > \epsilon, \;\; j = 1, \ldots, n$.

Using the exact bound presented above when possible, or otherwise the bounds presented in Section 4.1.2, we can apply the BSSS method to solve the (planar) BSPL problem. In the next section we adapt the BSSS method also to solve the (network) BSNL problem.

4.2 The network case : The BSNL problem

In this section we adapt the BSSS method to the network case. However, instead of dividing big squares into smaller squares, we divide edges into sub-edges. This will be explained in detail in Section 4.2.1. Assume we have an undirected connected network $G(\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ where $|\mathcal{V}| = n$ nodes, and a finite set of edges (arcs) $\mathcal{E} = \{(v_i, v_j), (v_k, v_l), \ldots, (v_p, v_q)\}$ with $|\mathcal{E}| = m$. Edges may also be denoted by e. All edges have a strictly positive length. Each node v_j carries two non-negative weights (w_j^1, w_j^2), one for the obnoxious criterion and one for the desirable criterion.
The model is the same as (7), except that the set of possible new locations is the entire network. With our choice of obnoxious objective function, however, x cannot be located in a node. Therefore, we again require, that $d(x, v_j) > \epsilon, \;\; j = 1, \ldots, n$, where ϵ is a small positive number. The BSNL problem is then:

$$\begin{array}{ll} \min & f(x) = \sum\limits_j \; w_j^1 \; (d(x, v_j))^{-b}, \qquad b > 0 \\ \min & g(x) = \sum\limits_j \; w_j^2 \; d(x, v_j) \\ \text{s.t.} & \\ & x \in G(\mathcal{V}, \mathcal{E}) \end{array} \tag{8}$$

where $d(x, v_j)$ is the shortest distance from point x to node v_j. The authors are well aware that the obnoxious objective function is not as appropriate on the network model, as in the planar model, but we have decided to use it for comparison purposes, see Paper C. The approximation algorithm is a very general and intuitive approach and can be used for complicated objective functions.

4.2.1 The Edge Dividing (ED) algorithm

The idea of the Edge Dividing (ED) algorithm is similar to the idea behind the BSSS algorithm. First we divide each edge into two subedges. Then bounds on the objective function values on each subedge are calculated. Furthermore, a sample set of objective function values are calculated. If the bounds calculated for a subedge are dominated by one (or more) of the sample set objective function values, then the subedge is dominated and may be deleted from further consideration.
The bounds are derived in detail in Sections 4.2.2 and 4.2.3. The sample set of objective function values are calculated in the middle (center) of the subedges. The nondominated criterion values are kept in a list.

The output from the algorithm is an ordered set of "efficient" subedges. This general procedure, however, has a few disadvantages. The efficient set (or part of it) may be an edge-segment. This subedge will obviously remain efficient, but the subedge will be divided into smaller subedges again and again. This reveals that the list of efficient subedges will probably almost double in size, when we half the ϵ value, for ϵ sufficiently small. This can in fact be used as an alternative stopping criterion.

4.2.2 Calculating lower bounds

We need both upper and lower bounds on the distance $d(x, v_j)$, where x can be any point on the edge (or sub-edge) e_i. We refer to the lower bound of this distance by $\underline{d(e_i, v_j)}$ and to the upper bound by $\overline{d(e_i, v_j)}$. Assume $e_i \in (v_h, v_k)$, and x_h is the endpoint of e_i closest to v_h, and that x_k is the endpoint of e_i closest to v_k.
The upper bound may be calculated as

$$\overline{d(e_i, v_j)} = \min\{d(v_j, v_h) + d(v_h, x_h), d(v_j, v_k) + d(v_k, x_k)\} + d(x_h, x_k)$$

and the lower bound may be calculated as

$$\underline{d(e_i, v_j)} = \min\{d(v_j, v_h) + d(v_h, x_h), d(v_j, v_k) + d(v_k, x_k)\}.$$

These two bounds can easily be calculated as illustrated in Figure 11, whenever the distance matrix D, of shortest distances between all pairs of nodes, is available. A procedure to obtain D can be found in Thulasiraman and Swamy [71].

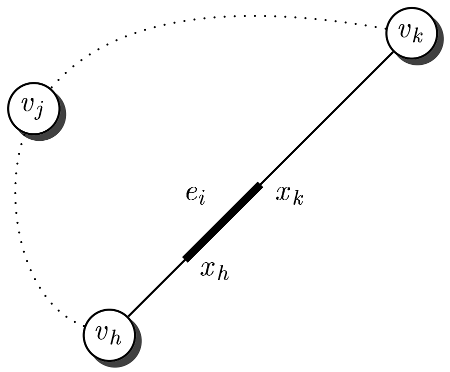

Figure 11: Calculating distance bounds.

Using these bounds, we can calculate the lower bounds on the objective function values as

$$\begin{aligned} \underline{f}(e_i) &= \sum_j w_j^1 \left(\overline{d(e_i,v_j)}\right)^{-b} \\ \underline{g}(e_i) &= \sum_j w_j^2 \, \underline{d(e_i,v_j)} \end{aligned}$$

4.2.3 Exact bounds

In this section we derive some exact bounds, specifically for our choice of objective functions. The distance function $d(x, v_j)$ is a concave function on an edge (subedge). Therefore $g(x)$ is a concave function on an edge, and the minimum is always in one of the (sub-edge) endpoints. So we have an exact lower bound as follows.

$$\underline{g}(e_i) = \min\{g(x_h), g(x_k)\} \tag{9}$$

Now, lets consider $f(x)$. Since $d(x, v_j)$ is both positive and concave, $(d(x, v_j))^{-b}$ is convex. Therefore $f(x)$ is convex on an edge. If we are looking at the sub-edge from x_h to x_k as illustrated in Figure 11, and the derivatives at the endpoints have the same sign, then an exact lower bound is simply the smallest endpoint value. That is, if

$$sign\left(\frac{\partial^+}{\partial x_{(v_h,v_k)}} f(x_h)\right) = sign\left(\frac{\partial^+}{\partial x_{(v_h,v_k)}} f(x_k)\right) \tag{10}$$

then

$$\underline{f}(e_i) = \min\{f(x_h), f(x_k)\} \tag{11}$$

where $\frac{\partial^+}{\partial x_{(v_i,v_j)}} f(x)$ denotes the derivative in the direction from v_i towards v_j, and we want to know if the function increases or decreases at x. The "+" indicates right derivative, so even in a break-point this derivative is well-defined. If (10) does not hold, the bound in Section 4.2.2 has to be applied. For more general objective functions, the bounds in Section 4.2.2 may be needed more often.

4.3 Comparison of the BSPL and the BSNL problems

Even though the planar and the network models may seem very different in structure, they are designed to solve the same real-life problem. Often a combination of the two models would be preferable. In [4] and [70] planar and network models are combined. Modeling air pollution such as noise makes most sense in the planar model, whereas the network model is a good description of a road network with distances or travel times as coefficients. One possible combination is to embed the network on top of the plane, so that each point on the network corresponds to a point in the plane, but not the other way round. This is illustrated in Figure 12.

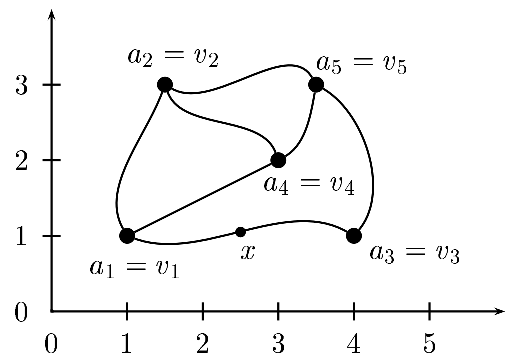

Figure 12: Combination of network and planar models.

The point x in Figure 12 is in the middle of edge (v_1, v_3), but it is also associated with a point in the plane, namely $(2.5, 1)$, which can be used in an l^p-norm calculation.
Which model is the most appropriate is not always easy to determine. The available data will probably decide the model.

4.4 Conclusions on approximation methods

In this section we have described a powerful tool for approximating the set of efficient solutions on both planar and network models. The method can be applied when exact solution methods are not at hand, as is often the case when the objective functions are nonlinear functions.
It can be seen that the two solution algorithms (presented in detail in Paper C), are not restricted to the bicriteria case. If the dominance check routine is adapted to the

multicriteria case, the same method can be used. The method does in fact approximate the efficient set of the multicriteria semi-obnoxious location problem.

In Paper C an example of both the planar and the network models are examined. The example involves the location of a new airport near the city of Aarhus, Denmark. The experimental results presented in the paper are quite satisfactory.

5 Multicriteria Semi-obnoxious Network Location (MSNL) problems

There are a number of models that deal with the problem of locating (placing) a new facility on a network. Most of these models locate a desirable facility, such as a supermarket or a fire station, where the objective is to keep the new facility close to its users. The two most common ones are the minisum and minimax (weighted median and weighted center). There are also some models describing how to locate an obnoxious (undesirable) facility, such as a nuclear power plant or a dump cite which the users want to locate far away. In obnoxious theory the two most common objective functions are the maxisum and maximin (weighted anti-median and weighted anti-center). Many facilities can, however, be thought of as semi-obnoxious. Such facilities could be airports, train stations or other noisy service facilities. It could also be the above-mentioned dump cite that, with respect to transportation costs, should be located centrally, but, in the neighbors' opinion, should be located distantly. These location problems could with obvious advantages be formulated as Multicriteria Semi-obnoxious Network Location (MSNL) problems. In this way the trade-off between the different objectives can be revealed, making a good basis for an overall decision. Different aspects of the problem can be described by different objectives. Such objectives could be transportation costs, travel time, air pollution or minimizing the number of citizens within a certain radius of the facility. Another situation arises when we have more decision makers, each having their own objective function. References to related litterature are found at the beginning of Section 4.

We have found no literature describing the MSNL problem, but a general solution method for the multicriteria median problem is presented in Hamacher *et al.* [36]. This problem involves only desirable sum objective functions, but we have generalized the method to work for more general models.

5.1 Problem formulation and definitions

We are given a (strongly) connected network $G(\mathcal{V},\mathcal{E})$ with nodeset $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ where $|\mathcal{V}| = n$ nodes, and edgeset $\mathcal{E} = \{(v_i, v_j), (v_k, v_l), \ldots, (v_p, v_q)\}$ with $|\mathcal{E}| = m$ edges. If the underlying graph is directed, it is denoted G_D, and the edge $e = (v_i, v_j)$ has head v_j and tail v_i. If the underlying graph is undirected it is just denoted G, and $e = (v_i, v_j) = (v_j, v_i)\ \forall e \in \mathcal{E}$. We define the set of objectives as $\mathcal{Q} = \{1, 2, \ldots, Q\}$. Each node v_i carries Q weights $(w_i^1, w_i^2, \ldots, w_i^Q)^t$, where $w_i^q > 0, \forall q \in \mathcal{Q}$, so we may refer to the matrix of weights by $W_{Q\times n}$. Each edge $e \in \mathcal{E}$ has length $l(e) \in \mathbb{R}_+$.

By $d(v_h, v_k)$ we denote the distance between v_h and v_k which is given by the length of a shortest path between v_h and v_k. A point $x \in G(\mathcal{V}, \mathcal{E})$ can be located both at a node or on an edge.

We define a **point** x on a directed edge $e = (v_i, v_j)$ as a touple $x = (e, t), t \in [0, 1]$, with

$$d(v_k, x) = d(v_k, v_i) + tl(e) \quad \text{and} \quad d(x, v_k) = (1-t)l(e) + d(v_j, v_k)$$

for any $v_k \in \mathcal{V}$. A **point** x on an undirected edge $e = (v_i, v_j)$ is defined as a touple $x = (e, t), t \in [0, 1]$, with

$$d(x, v_k) = \min\{d(v_k, v_i) + tl(e), d(v_k, v_j) + (1-t)l(e)\}$$

for any $v_k \in \mathcal{V}$. Notice that $d(v_i, x) = tl(e)$ and $d(x, v_j) = (1-t)l(e)$ for $x = (e, t)$. Since $v_i = (e, 0)$ and $v_j = (e, 1)$, all nodes in the network are also points in the network.

The set $\{(e, t) | t \in (t_1, t_2), t_1, t_2 \in [0, 1]\}$, forming an open **subedge** on e, is denoted $(e, (t_1, t_2))$ for any $e \in \mathcal{E}$. Of course this set is empty, unless $t_2 > t_1$. Similarly, we define closed and half right/left open subedges.

We formulate the model with the maxisum and minisum objectives, which are obviously negatively correlated. For the undirected problem the objective functions are defined by

$$f^q(x) = \sum_{i=1}^{n} w_i^q \, d(x, v_i) \quad q \in \mathcal{Q} \tag{12}$$

and for the directed case they are defined by

$$f^q(x) = \sum_{i=1}^{n} w_i^q \, (d(x, v_i) + d(v_i, x)) \quad q \in \mathcal{Q} \tag{13}$$

The problem is formulated as follows:

$$\begin{array}{lll} \max & f^q(x) & q \in \mathcal{Q}_1 \\ \min & f^q(x) & q \in \mathcal{Q}_2 \\ \text{s.t.} & & \\ & x \in G(\mathcal{V}, \mathcal{E}) & \end{array}$$

$\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$, where $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$. $\mathcal{Q}_1$ is the set of obnoxious objective functions, and $\mathcal{Q}_2$ is the set of desirable objective functions. At most one of the sets are allowed to be empty. If $\mathcal{Q}_1 = \emptyset$ we have the situation discussed in Hamacher, Labbé and Nickel [36]. $f(x) = (f^1(x), f^2(x), \ldots, f^Q(x))^t$.

For simplicity in the succeeding argumentation, we multiply all objective functions in $\mathcal{Q}_1$ by -1 in order to minimize instead of maximize. In the remaining part of the section, we assume that $w_i^q < 0,\ i = 1, 2, \ldots, n$ and $q \in \mathcal{Q}_1$, and $w_i^q > 0,\ i = 1, 2, \ldots, n$ and $q \in \mathcal{Q}_2$. This gives the following problem formulation:

$$\begin{array}{lll} \min & f^q(x) & q \in \mathcal{Q}_1 \\ \min & f^q(x) & q \in \mathcal{Q}_2 \\ \text{s.t.} & & \\ & x \in G(\mathcal{V}, \mathcal{E}) & \end{array} \tag{14}$$

In order to find the shortest distances between x and all the nodes, we need the **distance matrix** D of shortest distances between all pairs of nodes. Note that $D_{ij} = d(v_i, v_j)$. This matrix can be calculated in $O(n^3)$ running time using Floyd's algorithm or by applying Dijkstra's algorithm to all n nodes. For details on these graph procedures, see Thulasiraman and Swamy [71]. For an undirected network the distance matrix D is symmetric.
We will now outline the concept of bottleneck-points as it is presented in Church and Garfinkel [15]. There are two types of bottleneck-points, edge-bottleneck-points and node-bottleneck-points. Only edge-bottleneck-points are defined here, because the nodes will be examined anyway, whether they are bottleneck-points or not. The edge-bottleneck-points are defined as follows, for each edge $(v_i, v_j) \in \mathcal{E}$: Let $x = (e, t)$ be on the edge (v_i, v_j). If there exists a node $v_k \neq v_i, v_j$ such that

$$D_{ki} + tl(e) = D_{kj} + (1 - t)l(e)$$

then x is an **edge-bottleneck-point**. It is easily seen, that edge (v_i, v_j) contains an edge-bottleneck-point with respect to node v_k if and only if

$$|D_{ki} - D_{kj}| < l((v_i, v_j))$$

This sets the upper bound on the number of edge-bottleneck-points on an edge to n. If we consider the endnodes of the edges as bottleneck-points as well, we have $O(n)$ bottleneck-points per edge. This gives a total of $O(mn)$ bottleneck-points on $G(\mathcal{V}, \mathcal{E})$.
We will denote the **edge-bottleneck-point matrix** of shortest distances from all edge-bottleneck-points to all nodes by B. So B_{ij} is the shortest distance from edge-bottleneck-point B_i to node v_j. This matrix is needed for easy calculation of the objective-values in the bottleneck-points. When we know the shortest distance matrix D, the bottleneck-points can be calculated in $O(n\ log\ n)$ time, see Hansen *et al.* [42].

The weighted-sum objective with positive weights is a piecewise linear, **concave** function on an edge, with break-points only in the edge-bottleneck-points, see Figure 13. If all weights are negative the objective function is a piecewise linear, **convex** function with break-points only in the edge-bottleneck-points.

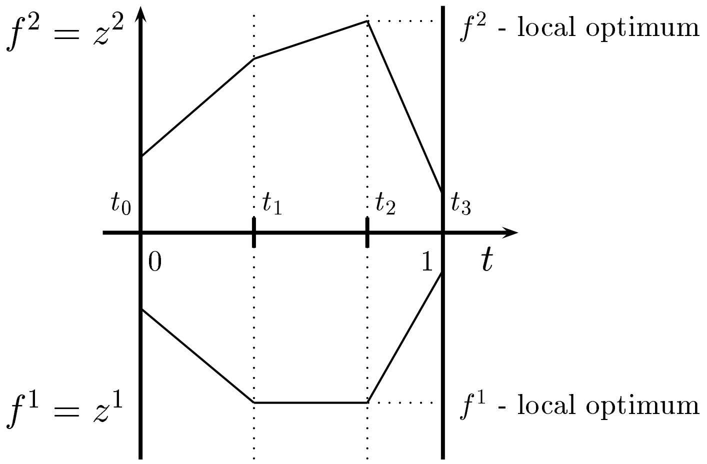

Figure 13: Illustration of the objective functions on an edge.

Note that the optimum for the maxisum need not be unique, it can be a subedge between two (or more) bottleneck-points, or there may be points on different edges with the optimal value. The optimum for the minisum is attained at one or more nodes.

5.1.1 Example

Now we present a small example to illustrate the structure of the undirected problem, see Figure 14. Let the weights be $w^1 = (-1, -2, -1, -1, -2, -2)$ and $w^2 = (2, 1, 2, 2, 2, 1)$. Let the distance matrix D be given by

$$D = \begin{bmatrix} 0 & 1 & 1 & 4 & 3 & 2 \\ 1 & 0 & 2 & 3 & 4 & 1 \\ 1 & 2 & 0 & 3 & 2 & 3 \\ 4 & 3 & 3 & 0 & 5 & 2 \\ 3 & 4 & 2 & 5 & 0 & 3 \\ 2 & 1 & 3 & 2 & 3 & 0 \end{bmatrix}$$

for the undirected network of Figure 14. B can be calculated as

$$B = \begin{bmatrix} 2 & 3 & 3 & 6 & 1 & 4 \\ 3 & 2 & 4 & 1 & 6 & 3 \\ 2 & 3 & 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 & 4 & 3 \\ 2 & 3 & 1 & 4 & 1 & 4 \\ 3 & 2 & 4 & 1 & 4 & 1 \\ 4 & 3 & 3 & 4 & 1 & 2 \\ 3 & 2 & 4 & 3 & 2 & 1 \end{bmatrix}.$$

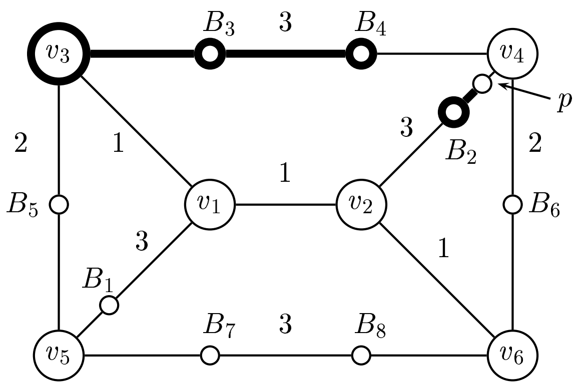

Figure 14: The undirected network of Example 5.1.1. The bold parts constitute the set of efficient points.

To clarify the solution to the undirected network in Figure 14 we present some objective function values in Table 6. A solution method specifically for the bicriterion model is described in Section 5.3. The general multicriteria method is described in Section 5.2.3. Please note the values of p and B_4. This proves that a subedge, not having endpoint at a node or a bottleneck-point, can be efficient. We will refer to this example in Section 5.2 and 5.3. From Table 6 we note that bottleneck-point B_2 is optimal for the maxisum criterion (f^1) and node v_3 is optimal for the minisum criterion (f^2).

Point x	$f(x) = (f^1(x), f^2(x))$
v_1	$(-17, 19)$
v_2	$(-16, 21)$
v_3	$(-18, 17)$
v_4	$(-27, 29)$
v_5	$(-24, 27)$
v_6	$(-15, 21)$
B_1	$(-27, 31)$
B_2	$(-30, 33)$
B_3	$(-25, 23)$
B_4	$(-28, 27)$
B_5	$(-23, 29)$
B_6	$(-20, 27)$
B_7	$(-25, 25)$
B_8	$(-23, 27)$
p	$(-28, 30\frac{1}{3})$

Table 6: Criterion values for all nodes, all bottleneck-points and point p.

5.2 General solution method for the Q criteria case

First, we solve two simple cases of the problem, namely the node problem and the directed case of the absolute location problem. Then we present the absolute location problem on an undirected network.

5.2.1 Locating the new facility in a node

In this case the new facility can be placed only at the nodes of the given network, and we can determine the efficient set $\mathcal{X}_{Par} = \mathcal{X}_{Par}(\mathcal{V})$ by the following approach in $O(Qn^2)$ time, given the distance matrix D.

In the solution procedure we make a pairwise comparison of all the n nodes. Initially we classify all nodes as efficient. Then we compare, say nodes v_i and v_j. If $f(v_j)$ dominates $f(v_i)$, we delete v_i from the set of efficient nodes and continue the comparison. This approach is presented in both Hamacher *et al.* [36] and Paper D.

5.2.2 Locating the new facility on a directed network

For this problem we have to investigate the objective function (13) of the directed case. First we observe that the objective functions are constant on the interior of the edges. This result is proven in Paper D, and follows from the fact that each term in the sum in (13) is a shortest cycle multiplied by a weight.

We subsequently use the triangular inequality to prove that the obnoxious objective functions, $q \in \mathcal{Q}_1$, have a higher value at the endnodes of e, and that the desirable objective functions, $q \in \mathcal{Q}_2$, have a lower value at the endnodes of e. Thus, f^q is still convex for $q \in \mathcal{Q}_1$ and concave for $q \in \mathcal{Q}_2$ on an edge. To see this we analyze the objective function (13) once again. This result is also proven in Paper D. This structure of the objective functions on the directed edges is illustrated in Figure 15. The values are taken from a directed example presented in Paper D.

The solution procedure for the directed case is very similar to the procedure for the "node" case in Section 5.2.1. When we have observed that the objective functions are constant on the interior of all edges, we can simply make a pairwise comparison of all nodes and edges. When we make this comparison on the $n+m$ nodes and edges, each taking $O(Q)$ time, we get a bound of $O(Q(n+m)^2)$ time.

5.2.3 Locating the new facility on an undirected network

The general solution method consists of pairwise comparison of subedges. The objective functions are all piecewise linear, and the idea is to partition the network into subedges,

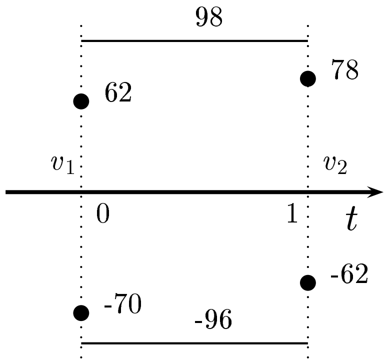

Figure 15: $f((v_1, v_2))$. Notice that $f(v_1)$ dominates $f(v_2)$.

where the objective functions are linear. The points where the piecewise linear functions change in slope are in fact the bottleneck-points. We then make a pairwise comparison of all these subedges, and delete the inefficient parts. The result is the complete set of efficient solutions $\mathcal{X}_{Par}$. For each comparison of two subedges we will construct a linear program, that can be solved in **linear time** by methods found in Megiddo [53], to detect inefficient points.

Let $z^q(t) = f^q(x_t),\ x_t = (e,t),\ e = (v_i, v_j)$. These Q functions are all piecewise linear, with the same set of possible breakpoints corresponding to the bottleneck-points. Assume there are $P+1$ breakpoints, including the two nodes. We then have P subedges on e. Let these breakpoints on (e,t) be denoted by $t_j,\ j = 0, 1, \ldots, P,\ (1 \le P \le n+1)$, with $t_0 = v_i,\ t_P = v_j$ and $t_{j-1} < t_j\ \forall\, j = 1, 2, \ldots, P$. For $t \in [t_{j-1}, t_j]$, the $z^q(t)$'s are linear functions of the form

$$z^q(t) = m_j^q t + b_j^q \quad \forall\, q = 1, 2, \ldots, Q.$$

Let us now compare the subedge A on edge e_A, $(e_A, [t_{j-1}, t_j])$, with subedge B on edge e_B, $(e_B, [s_{p-1}, s_p])$. A point $(e_A, t) \in (e_A, [t_{j-1}, t_j])$ is dominated by some point $(e_B, s) \in (e_B, [s_{p-1}, s_p])$ if and only if

$$m_p^q s + b_p^q \le m_j^q t + b_j^q \quad \forall\, q = 1, 2, \ldots, Q$$

where at least one inequality is strict. This comparison is illustrated in Figure 16 for two subedges from Example 5.1.1. Subedge (B_7, B_8) is compared with subedge (v_5, B_7).

Let us define the set $\mathcal{T}$ where the inequalities hold (for these particular subedges) by

$$\mathcal{T} = \{(s,t) |\ m_j^q t - m_p^q s \ge b_p^q - b_j^q,\ \forall\, q \in \mathcal{Q}\} \cap ([s_{p-1}, s_p] \times [t_{j-1}, t_j])$$

If $\mathcal{T} = \emptyset$, $(e_B, [s_{p-1}, s_p])$ does not contain a point dominating any point in $(e_A, [t_{j-1}, t_j])$. Otherwise $\mathcal{T} \neq \emptyset$ is taken as a feasible solution set of the two 2-variable linear programs:

$$LB = \min\{\ t \mid (s,t) \in \mathcal{T}\} \quad \text{and} \quad UB = \max\{\ t \mid (s,t) \in \mathcal{T}\}$$

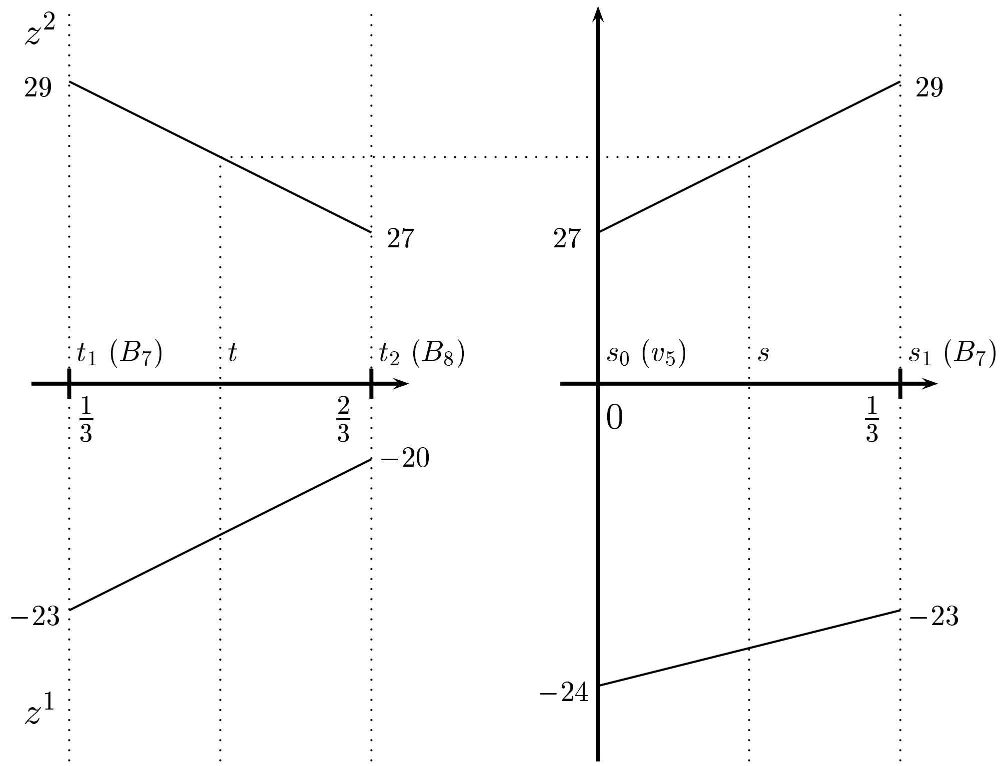

Figure 16: Comparing subedge (B_7, B_8) with subedge (v_5, B_7).

Using methods described by Megiddo [53], LB and UB can be calculated in $O(Q)$ time. Next we determine if the points corresponding to LB and UB are only weakly dominated. This means that none of the inequalities need to be strict as required by Definition 1. The details of this are found in Paper D. If both LB and UB are dominated, we delete the dominated part of $(e_A, [t_{j-1}, t_j])$ as follows:

$$(e_A, [t_{j-1}, t_j]) = (e_A, [t_{j-1}, t_j]) \setminus (e_A, [LB, UB])$$

This subedge comparison is illustrated in Figure 17, where the subedge $(B_7, B_8) = (e, [\frac{1}{3}, \frac{2}{3}])$ from Example 5.1.1 is compared with $(v_5, B_7) = (e, [0, \frac{1}{3}])$. Both subedges are on the same edge. Since $\mathcal{T}$ is non-empty we solve the two programs and find $LB = \frac{1}{3}$ and $UB = \frac{2}{3}$. Both LB and UB are dominated, so the subedge (B_7, B_8) is completely deleted.
In order to complete the comparison, we simply make an ordered subedge comparison. First, we compare $(e_1, [t_0, t_1])$ with all the other subedges, possibly deleting parts of $(e_1, [t_0, t_1])$. Then we compare the second subedge $(e_1, [t_1, t_2])$ with all the remaining subedges, including the subedge $(e_1, [t_0, t_1])$. This comparison continues until we have compared the last subedge $(e_m, [s_{P-1}, s_P])$ with all the remaining subedges.

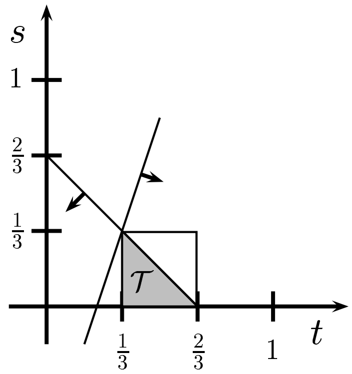

Figure 17: The linear programming constraints for comparing $(B_7, B_8) = (e, [\frac{1}{3}, \frac{2}{3}])$ with $(v_5, B_7) = (e, [0, \frac{1}{3}])$ on edge (v_5, v_6) in Example 5.1.1. $\mathcal{T}$ is indicated by the shaded area.

Notice that we can still use the entire subedge $(e_A, [t_{j-1}, t_j])$ to compare with the other subedges, even though a part of it is inefficient. It is only for the set of efficient points $\mathcal{X}_{Par}$, that we have to remember what part of $(e_A, [t_{j-1}, t_j])$ which is efficient.
If we make the global pairwise comparison on the $O(mn)$ bottleneck-point subedges, each taking $O(Q)$ time, we get a complexity bound of $O(Qm^2n^2)$ time. This is also the bound for the case where $\mathcal{Q} = \mathcal{Q}_2$ found in Hamacher *et al.* [36].

5.3 Bicriteria case

In this section we present an improved method for the 2-criteria case. When we have only two criteria, we may use the image of the network mapped into criterion space $\mathcal{Z}$ to solve the problem faster. This is done by calculating the lower envelope, see Hershberger [45]. The envelope can be calculated in $O(p\ log\ p)$ time, where p is the number of line-segments (subedges).
This procedure is best described by an example, so we present the undirected network of Example 5.1.1 in criterion space, see Figure 18.
Since we want to find the set of efficient solutions $\mathcal{X}_{Par}$, we are only interested in values between the two extreme optimal solutions, namely f^{1*} and f^{2*}. In criterion space we are only interested in the region $[f^{1*}, f^1(x^2)] \times [f^{2*}, f^2(x^1)]$, where x^1 and x^2 are defined in Section 1.1. We have to make sure that the slope of the envelope is decreasing, when the f^1-values increase, to ensure that there are no dominated points on the envelope. This can easily be ensured by a few technical details described in Paper D. The lower envelope now constitutes $\mathcal{Z}_{Par}$. The set of efficient solutions is then given by $\mathcal{X}_{Par} = f^{-1}(\mathcal{Z}_{Par})$. The efficient set corresponding to the nondominated set of Figure 18 is indicated in Figure 14. We have the same complexity bound on the lower envelope calculation as in Hamacher *et*

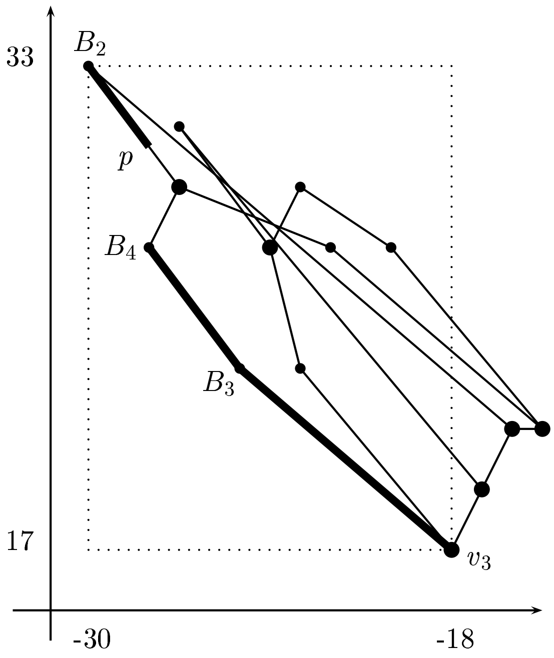

Figure 18: Mapping of the undirected network from Example 5.1.1 into criterion space. The bold parts constitute the set of nondominated points.

al. [36], namely $O(mn\ log(mn))$. This bound can be rewritten by examining the log term and using the fact that m is at worst n^2 for dense graphs. We therefore get the bound of $O(mn\ log\ n)$ time for the envelope calculation.

5.4 Computational results

In this section we present computational results from an implementation of the algorithm outlined in Section 5.2.3. We have not used the methods of Megiddo [53] in this implementation to solve the small LP's. Instead, we have used CPLEX 6.6. The code is programmed in C++ and the tests are run on a 700 MHz Linux PC.
We have used random networks of varying size generated using NETMAKER, see Paper B [66]. In each group we have used 10 random networks, and the mean is reported in the following tables.

First, we examine some semi-obnoxious bicriterion networks, having one push objective and one pull objective. The results are presented in Table 7. It appears that the number of subedges grows a little less than squared the number of nodes. The number of actual comparisons made is presented in the table, and the percentage of actual comparisons to the worst case is also presented. It is important to note that this percentage decreases as the networks increase in size.
The number of efficient subedges is also presented in Table 7, and this number seems to

# Nodes	50	100	150	200	250
CPU-time	40.96	229.54	774.64	1505.42	3326.37
# Subedges	3033.6	9411.5	18525.2	28368.1	39540.2
# Subedge comparisons (in millions)	0.358	1.770	5.138	8.655	16.531
# Efficient subedges	96.2	155.3	175.7	222.5	264.5
% Efficient subedges	3	1.6	0.95	0.78	0.67
% Comparisons	4.00	2.02	1.50	1.08	1.05
# Comparisons per sec	8733	7709	6633	5749	4970

Table 7: Semi-obnoxious bicriterion results, 1 push - 1 pull objective.

grow linearly with the number of nodes. This number is in fact higher than the number of actual efficient subedges, because more subedges may contain the same efficient point, when this point is a node. If a node is efficient, all the subedges connected to this node contain some efficient points (perhaps only the node which is the endpoint of the subedge). The last row in Table 7 are the numbers of comparisons made per CPU-second. Assuming that CPLEX performs independently of the number of problems it has to solve, this decrease indicates that the large problems require a lot more storage of data, and accessing this data takes an increasing amount of time.

Next we examine the effect of having more objectives. These results are all computed on networks with 50 nodes. We reuse the results of the bicriterion (1-1) networks of Table 7, examine two types of three objective problems and one type of four objective problems. The three objective networks are generated with both 1 obnoxious and 2 desirable objectives (1-2), and 2 obnoxious and 1 desirable objectives (2-1). The four objective networks are all with 2 obnoxious and 2 desirable objective functions (2-2). The results are presented in Table 8.

As expected both the number of subedges containing efficient points and the CPU-time increase rapidly when more negatively correlated objective functions are added. With four objectives more than 75 % of the subedges contain efficient points. It is seen that the CPU-time for these instances is almost proportional to the number of subedge comparisons, since the data size of the instances is approximately the same (last line in Table 8).

Finally, we conclude that the computational results are constructive in the sense that rather large problems can be solved within a reasonable amount of time. Since location problems are usually not of the type you have to resolve often, a longer CPU-time is acceptable.

The most encouraging result being that for bicriterion networks with objective functions in almost opposite directions, a very small proportion of the networks is efficient. This

# Objectives	1-1	1-2	2-1	2-2
CPU-time	40.96	123.05	105.49	870.57
# Subedges	3033.6	3293.1	3158.8	2853.6
# Subedge comparisons (in millions)	0.358	1.019	0.914	6.128
# Efficient subedges	96.2	359.1	357.9	2237.7
% Efficient subedges	3	11	11	78
% Comparisons	4.00	9.47	9.53	75.46
# Comparisons per sec	8733	8349	8720	7077

Table 8: The effect of having more objectives. All networks have 50 nodes.

indicates that this model is in fact an aid for the decision-maker, since a large part of the network can be omitted from further consideration. On the efficient parts of the network, the trade-off between the two objectives can then be revealed.

As a final comment, we note that with negatively correlated objectives, at most three objective functions should be considered. Otherwise, a very large proportion of the network will be efficient, and this method will not have helped the DM.

5.5 Conclusions on the subedge comparison approach

After having investigated the different problems in turn, we can conclude that the methods described in Section 5.2.3 and 5.3 works for any piecewise linear objective function. The effectiveness of the two algorithms depend on how easy the breakpoints can be found, and on the number of resulting subedges. If E is the number of subedges, the bicriterion method from Section 5.3 runs in $O(E\ logE)$ time and the multicriteria method from Section 5.2.3 runs in $O(QE^2)$ time.

From the section on computational results we can conclude that the method is applicable the problems of a fair size. We have also seen that for biobjective problems with negatively correlated objective functions only a limited part of the network is efficient. We therefore conclude that this model is in fact a good tool for MSNL problems.

6 Bicriteria Network Location (BNL) problems with criteria dependent lengths and minisum objectives

We begin this section by a motivating example. Assume we have to locate a money reserve, considering the two objectives of minimizing the transportation costs and the risk of having the transports robbed. The depot serves a number of clients varying in size, and we are given a connected network and interpret each of the n nodes as the clients. A relevant (node) weight for a client with respect to transportation costs is the number of monthly deliveries, and a weight for the risk objective is the maximum value of a money-transport. The edge-lengths with respect to transportation costs could be the distance, and for the risk objective the edge-length could be the probability of an assault. If we assume that the cost of opening the new facility is independent of location, this particular cost is unimportant.

A solution to this problem consists of two decisions. The first (and probably the most important) one is to decide where to locate the new facility (depot), and the second one consists in determining how to route the flow from the new facility to the nodes. The flow problem consists of $n-1$ Bicriterion Shortest Path (BSP) problems (described in Section 2).

The solution method proposed is a variant of the two-phases approach due to Ulungu and Teghem [74] and Visée *et al.* [75]. In Phase 1 all (or a representative subset of) the supported extreme solutions are found by using the weighting method. In Phase 2 a search between the supported solutions is conducted to find unsupported efficient solutions. The procedure is explained in details in Section 6.3.

6.1 Problem formulation

We are given a connected directed network $G(\mathcal{V},\mathcal{E})$ with node set $\mathcal{V}=\{v_1,v_2,\dots,v_n\}$ where $|\mathcal{V}|=n$ nodes, and edge set $\mathcal{E}=\{(v_i,v_j),(v_k,v_l),\dots,(v_p,v_q)\}$ with $|\mathcal{E}|=m$ edges. The underlying graph is denoted by G, and edges may be referred to by $e\in\mathcal{E}$, by $(v_i,v_j)\in\mathcal{E}$ or simply by $(i,j)\in\mathcal{E}$, where node i is the tail and node j is the head. Each node v_i carries two weights $(w_i^1,w_i^2)^t$, where $w_i^q\in\mathbb{R}_+,\ q=1,2$, so we may refer to the matrix of weights by $W_{2\times n}$. Each edge $e\in\mathcal{E}$ has length $l(e)=(l^1(e),l^2(e))\in\mathbb{R}_+^2$. Let us define a matrix of edges $E_{m\times(4)}$ with the following entries. E_{i1} is the tail of edge e_i, E_{i2} is the head, $E_{i3}=l^1(e_i)$ is the length with respect to criteria one and $E_{i4}=l^2(e_i)$ is the length with respect to criteria two.

Notice that an undirected network can be modeled as a directed network with the double amount of edges. Define binary decision variables as follows:

$$x_i = \begin{cases} 1 & \text{if the facility is located in node } i \\ 0 & \text{else} \end{cases}$$

$$y_{ijk} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is used in the path to node } k \\ 0 & \text{else} \end{cases}$$

We examine the so-called median objectives or weighted sum objectives:

$$f^q(y) = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} l_{ij}^q w_k^q y_{ijk} \qquad q = 1, 2$$

Combining the coefficients to $c_{ijk}^q = l_{ij}^q w_k^q$, we get

$$f^q(y) = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijk}^q y_{ijk} \qquad q = 1, 2 \tag{15}$$

There are two types of constraints. The first constraint ensures that exactly one facility is located and the second set of constraints ensures the existence of paths from the facility to the remaining nodes. This leads to the following problem:

$$\begin{aligned} &\min \quad f^1(y) \\ &\min \quad f^2(y) \\ &\text{s.t.} \\ &\qquad \sum_{i=1}^{n} x_i = 1 \\ &\qquad \sum_{j=1}^{n} y_{jik} - \sum_{j=1}^{n} y_{ijk} = -x_i \qquad i \neq k, \quad \forall\, i, k \\ &\qquad x_i \in \{0,1\} \qquad \forall i \\ &\qquad y_{ijk} \in \{0,1\} \qquad \forall\, i, j, k \end{aligned} \tag{16}$$

Notice that we have omitted the following redundant constraints

$$\sum_{j=1}^{n} y_{jik} - \sum_{j=1}^{n} y_{ijk} = 1 - x_i \quad \forall\, i, \text{ where } i = k.$$

The reason being that this part of the constraint matrix consists of n totally unimodular sub-matrices forming the n sets of paths. Notice that one path is non-existing, since the node in which the new facility is located, ships nothing through the network to itself.

In Paper E we prove by an example that the constraint matrix of (16) is **not** totally unimodular.

Weighting the two objective functions in (16), using the weights λ and $1-\lambda$, $\lambda \in (0;1)$, results in the weighted version of (16)

$$\begin{array}{lrcll} \min & \lambda f^1(y) + (1-\lambda) f^2(y) & & & \\ \text{s.t.} & & & & \\ & \sum_{i=1}^{n} x_i & = & 1 & \\ & \sum_{j=1}^{n} y_{jik} - \sum_{j=1}^{n} y_{ijk} & = & -x_i & i \neq k \quad \forall\, i,k \\ & x_i \in \{0,1\} & & \forall i & \\ & y_{ijk} \in \{0,1\} & & \forall\, i,j,k & \end{array} \tag{17}$$

In Section 6.3.1 we describe how problem (17) can be solved in $O(n^4)$ running time using Benders' decomposition for a fixed λ. The appropriate λ is found as described by the NISE method, very similar to Phase 1 in Section 3.2.

6.2 Example

We examine the network presented in Figure 19 with the following weights and undirected edges. Each column of W consists of the two node-weights.

$$W = \begin{bmatrix} 200 & 300 & 500 & 100 & 400 & 500 & 400 \\ 7 & 4 & 2 & 6 & 6 & 2 & 8 \end{bmatrix}$$

The first two columns of E are the tail and head nodes. The next two columns are the two edge-lengths.

$$E = \begin{bmatrix} 1 & 2 & 78 & 22 \\ 1 & 3 & 24 & 72 \\ 1 & 4 & 26 & 71 \\ 1 & 5 & 13 & 71 \\ 1 & 7 & 86 & 12 \\ 2 & 3 & 98 & 29 \\ 2 & 5 & 17 & 90 \\ 3 & 5 & 29 & 97 \\ 3 & 6 & 87 & 28 \\ 3 & 7 & 7 & 69 \\ 4 & 5 & 4 & 77 \\ 4 & 7 & 89 & 5 \\ 5 & 6 & 17 & 92 \\ 5 & 7 & 40 & 74 \\ 6 & 7 & 69 & 12 \end{bmatrix}$$

The resulting 11 nondominated criterion vectors are presented in Table 9. These criterion vectors are visualized in Figure 20 and it is seen that there are 6 supported and 5

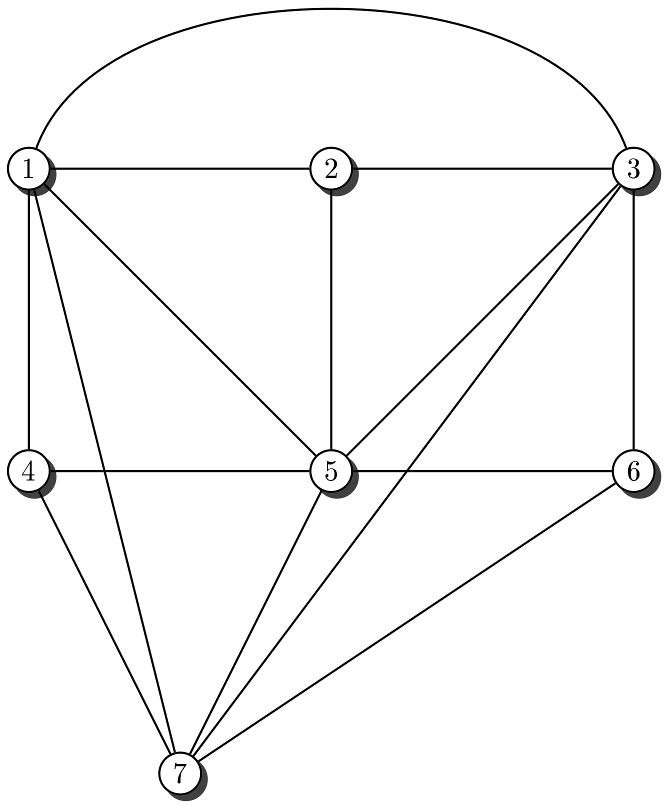

Figure 19: Network for Example 6.2.

unsupported criterion vectors. Of the 5 unsupported solutions, only one, $(89200, 1868)$, is locally unsupported (defined in Section 6.3). The other 4 unsupported solutions are locally supported by the nodes indicated in Figure 20. The last nondominated solution, $(89200, 1868)$, is dominated by a convex combination of the following two locally supported solutions:

$$\frac{9}{11}(91200, 1684) + \frac{2}{11}(80200, 2587) = (89200, 1848.18)$$

There are a total of 2128 feasible criterion vectors, using only efficient paths between nodes. All these vectors are illustrated in Figure 21.

6.3 Two-phases approach

In this section the solution procedure for solving the bicriterion problem (16) is outlined. Before stating the procedure it may be helpful to consider a naïve method. One possible way of solving the problem could be to solve problem (17) n times, namely one time for each possible location of the new facility. Suppose that the location of the new facility is fixed at a specific node, say node i (so $x_i = 1$). Using the weighting method, the supported efficient solutions (paths) with respect to node i can be revealed. We call these efficient solutions *locally* efficient (with respect to node i). Given $\lambda \in (0, 1)$ and x the corresponding

Node	f^1	f^2
5	45500	3025
5	47100	2289
1	78200	2062
7	89200	1868
7	91200	1684
1	92600	1506
7	97200	1376
1	107500	1182
7	111600	1112
7	129300	856
7	203800	798

Table 9: Nondominated values for Example 6.2.

locally efficient solution can be found in $O(n^3)$ running time, since we are faced with $n-1$ shortest path problems.

Finding the locally unsupported efficient solutions that are in fact globally efficient, constitutes a more difficult problem. These cannot be found using the weighting method. This fact is known from studying the BSP problem alone (Paper B [66]).

We thus have three types of efficient solutions:

- supported efficient solutions
- locally supported efficient solutions
- (locally) unsupported efficient solutions

The reason why locally supported efficient solutions are interesting, is that they may be unsupported efficient solutions in the main problem (16), but possible to find by the weighting method. These three kinds of solutions are illustrated in Figure 20.

The procedure that we propose instead of the naïve method, is a variant of the two-phases approach due to Ulungu and Teghem [74] and Visée *et al.* [75], and may be stated generically as:

- **Phase 1**: Find all (or a representative subset of) the supported solutions.
- **Phase 2**: Conduct a search between the supported solutions in order to find unsupported nondominated solutions.

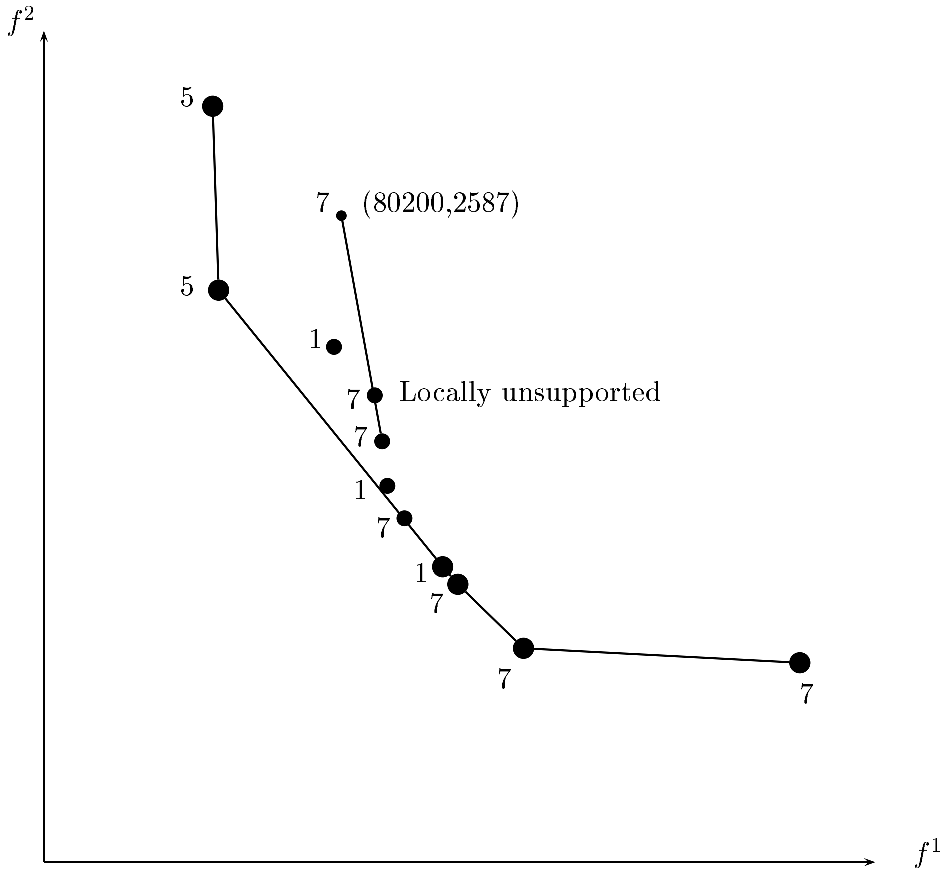

Figure 20: Nondominated vectors for Example 6.2. Large dots illustrate the supported solutions, and only one solution is locally unsupported. The numbers indicate the location node.

6.3.1 Benders' decomposition in Phase 1

As explained in Section 6.1 all supported solutions to (16) (and the locally supported) may be obtained by solving the weighted program (17) parametrically in $\lambda \in (0,1)$. We will do that by employing NISE (Non-Inferior Set Estimation), a method presented in Cohon [17]. NISE guides the choice of $\lambda \in (0,1)$. Details on how to compute the λ's are presented in Paper E.

Now we explain how Benders' decomposition can be used to find the supported solutions given a weight λ in Phase 1. Let λ be fixed and define

$$c_{ijk}(\lambda) = \lambda w_k^1 l_{ij}^1 + (1-\lambda) w_k^2 l_{ij}^2 \quad (\geq 0 \text{ since } l, w \geq 0).$$

When x is fixed, we can use the path constraints being totally unimodular, and relax the integrality constraints on y. Fixing x means locating the facility at a particular node. For

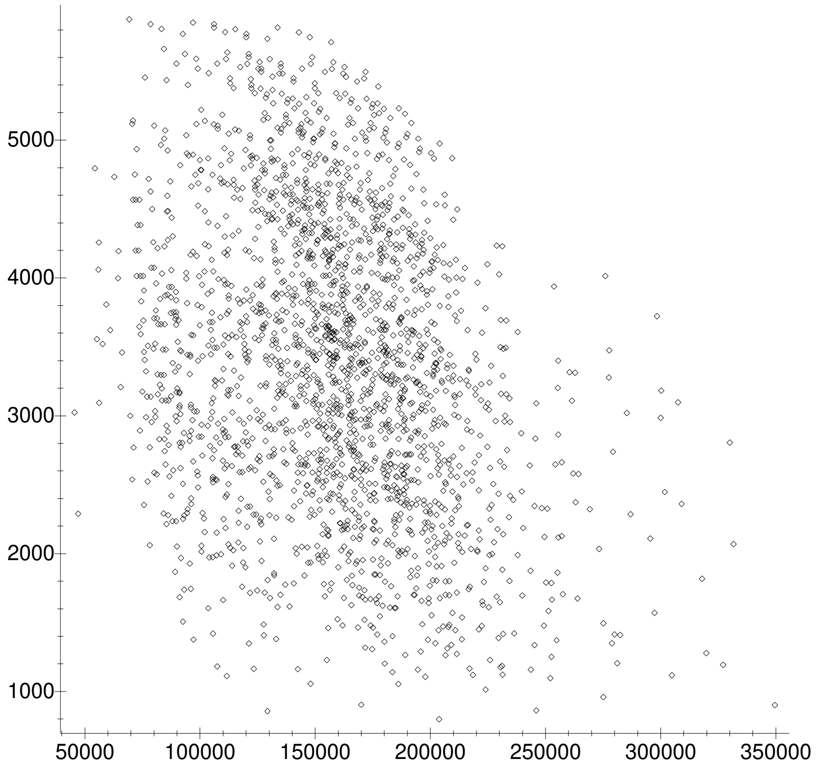

Figure 21: Illustration of 2128 criterion vectors for Example 6.2.

a fixed $\overline{x}$ satisfying $\sum_i x_i = 1,\ \ x_i \in \{0,1\}$, we get the following **Benders' subproblem**:

$$\begin{aligned} \min \quad & \sum_{k,i,j} c_{ijk}(\lambda) y_{ijk} \\ s.t \quad & \\ & \sum_j y_{jik} - \sum_j y_{ijk} \;=\; -\overline{x}_i \quad i \neq k \quad \forall i,k \\ & 0 \leq y_{ijk} \;\leq\; 1 \qquad \forall i,j,k \end{aligned} \tag{18}$$

This linear programming problem has the following dual program:

$$\begin{aligned} \max \quad & \sum_{\substack{i,k \\ k \neq i}} \alpha_{ik}(-\overline{x}_i) + \sum_{k,i,j} \beta_{ijk} \\ s.t \quad & \\ & \alpha_{jk} - \alpha_{ik} + \beta_{ijk} \;\leq\; c_{ijk}(\lambda) \quad i \neq k \quad \forall i,j,k \\ & \beta \;\leq\; 0 \end{aligned} \tag{19}$$

The variables α are free variables corresponding to the path constraints in (18) and the β variables correspond to the upper bound on y. These dual variables can be found when the $n-1$ shortest path problems are solved in the Benders' subproblem, so we need not actually solve the dual problem (19). The dual leads to the following **Benders' masterproblem**:

$$\begin{aligned}
\min \quad & v \\
s.t \quad & \\
& v \geq -\sum_{\substack{i,k \\ k\neq i}} \alpha_{ik}^{l} x_i + \sum_{k,i,j} \beta_{ijk}^{l} \quad \forall l \\
& \textstyle\sum_i x_i = 1 \\
& x_i \in \{0,1\} \quad \forall i
\end{aligned} \tag{20}$$

where l is an index for the added inequalities.

The first time we generate a redundant inequality (or suggests a node picked earlier), the solution at hand is optimal (efficient). This is true because the subproblem (18) will return an earlier found solution.

Notice that Benders' masterproblem (20) is easy to solve in this case. It can be reformulated as a minimax problem. Let us rewrite the first constraint in (20), keeping in mind that only one x_i will be one.

$$\begin{aligned}
v &\geq -\sum_i \sum_{\substack{k \\ k\neq i}} \alpha_{ik}^{l} x_i + \sum_{k,h,j} \beta_{hjk}^{l} \\
v &\geq \sum_i \left(-\sum_{\substack{k \\ k\neq i}} \alpha_{ik}^{l} + \sum_{k,h,j} \beta_{hjk}^{l} \right) x_i \\
v &\geq \sum_i c_i^l x_i \qquad \text{where } c_i^l = -\sum_{\substack{k \\ k\neq i}} \alpha_{ik}^{l} + \sum_{k,h,j} \beta_{hjk}^{l}
\end{aligned}$$

If we think of these c_i^l coefficients in a matrix, the optimal x_i is to find the column i where the largest c_i^l element is as small as possible.

Notice, that we have to solve problems (18) and (20) at most n times. Since Benders' subproblem consists of $n-1$ shortest path problems, problem (18) can be solved in $O(n^3)$ running time. Therefore the overall running time in Phase 1, given λ, is $O(n^4)$.

6.3.2 Phase 2

Here we can first find the locally supported nondominated vectors by using the weighting method for a fixed node(s).

To find locally unsupported efficient points of (16), we use the Tchebycheff theory. Let $z = (z^1, z^2)$ denote a fixed reference point with $z \leq z^* = \left(f^{1*}, f^{2*}\right)$, where z^* is the ideal point. Then the augmented non-weighted Tchebycheff program (21) may be stated as

$$\begin{array}{ll} \min & \alpha + \rho \left(f^1(y) + f^2(y) \right) \\ \text{s.t.} & \\ & \begin{array}{rcll} f^q(y) - \alpha & \leq & z^q & q = 1,2 \\ \sum_{i=1}^{n} x_i & = & 1 & \\ \sum_{j=1}^{n} y_{jik} - \sum_{j=1}^{n} y_{ijk} & = & -x_i & i \neq k \quad \forall i,k \\ x_i \in \{0,1\} & & \forall i & \\ y_{ijk} \in \{0,1\} & & \forall i,j,k & \\ \alpha \in \mathbb{R}_+ & & & \end{array} \end{array} \tag{21}$$

where ρ is a small positive constant ensuring that the solution is not just weakly efficient. A few comments are in order. Note that instead of solving the usual weighted Tchebycheff program as found in Steuer and Choo [69], we propose to solve the augmented non-weighted Tchebycheff program (21). It was shown by Alves and Climaco [1] that all nondominated solutions to (16) can be found using the non-weighted program for integer problems (IP), and in Alves and Climaco [2] this result was generalized to mixed integer problems (MIP). Note that the augmented Tchebycheff program (21) has the same constraints as our original problem (16), as well as two additional constraints. The two new constraints are the reference point constraints, linking the reference point to the objective function in (21). These two new constraints complicate the problem, since they destroy the nice structure of the constraint matrix. Using Lagrange relaxation of these constraints does not solve our problem. We simply end up with the weighting method. This is derived in Appendix 2 in Paper E. However, problem (21) is a one objective MIP, which can be solved by the usual IP methods, such as branch and bound.

Next we explain how to determine the appropriate reference point(s). Assume that we want to search for locally unsupported solutions between the two nondominated points E_1 and E_2. First, we determine a maximum deviation factor

$$\delta = \max \left\{ f^1(x^2) - f^{1*}, f^2(x^1) - f^{2*} \right\}$$

where x^1 and x^2 are defined in Section 1.1 as f^1 and f^2 optimal solutions. This deviation factor is going to ensure that our reference point is below the ideal point z^*. Next we find reference points corresponding to our two nondominated solutions, E_1 and E_2:

$$Z(E_i) = (E_i^1 - \delta, E_i^2 - \delta) \; i = 1,2$$

The search reference point z_{new} can then be determined as the maximum of the reference point coordinates, because this point has a maximum distance of δ to both $Z(E_1)$ and $Z(E_2)$:

$$z_{new} = \left(\max\left\{Z^1(E_1), Z^1(E_2)\right\}, \max\left\{Z^2(E_1), Z^2(E_2)\right\}\right).$$

Using z_{new} in (21) can result in two things. If a new solution is returned, this solution is nondominated and defines two new search areas. Otherwise one of the points E_1 or E_2 is returned, and no nondominated (unsupported) solutions exist between the two points.

For our Example 6.2 we find $\delta = \max\{203800 - 45500, 3025 - 798\} = 158300$. Next we search for locally unsupported solutions between the two points $E_1 = (78200, 2062)$ and $E_2 = (91200, 1684)$ (on either side of the single locally unsupported point in Figure 20). This leads to the reference point $z_{new} = (-67100, -156238)$, where $\alpha = 158300$ can find both E_1 and E_2. In this case $E_3 = (89200, 1868)$ is found with $\alpha = 158106$.

6.4 Conclusions on the BNL problem

We have presented a new, interesting location problem. The formulation incorporates both the location and the routing aspects in a multiobjective setting. We have also presented a solution method for the problem, and illustrate the problem structure and solution procedure by an example. The presented method can easily be made interactive, since the procedures in both phases are easily made interactive.

Unfortunately, the solution method does not easily generalize to more than two objectives. Difficulties exists in both phase 1 and 2 as explained in Paper E.

7 A stochastic programming model for capacity expansion at Sonofon

In this section we study a mobile communications network. Since the description of the problem in Paper F is very technical, we now make a less detailed presentation. This section is also quite different from previous sections, because the modeling phase of making a good description of a practical problem took a lot of effort. First, we will describe the structure of a mobile communications network.

The base transceiver stations (BTSs) are each connected to one base station controller (BSC). Each BSC serves a number of BTSs and is connected to one mobile switching center (MSC). Finally each MSC serves a number of BSCs and the MSCs are connected internally. The network is illustrated in Figure 22.

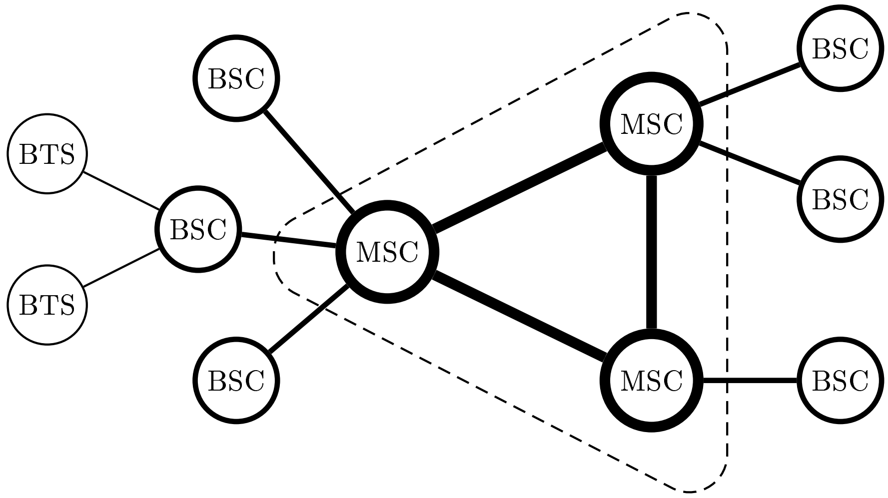

Figure 22: Illustration of mobile telecommunications network.

The visitor location register (VLR) of an MSC, a database handling all information about clients, has a limited capacity, thus restricting the number of customers that can be served (through BTSs and BSCs) by an MSC. Thus the network provider not only has to expand the link capacities but should consider when and where to deploy new MSCs in order to be able to serve the increasing number of customers.

We will consider the problem of deploying a number of new MSCs and allocating the BSCs to new and existing MSCs, thus treating the number and locations of BTSs and BSCs as exogenous. The deployment of MSCs must be done so as to minimize the incurred costs while meeting customer demand and observing the capacity restrictions. The cost function will include four terms:

1. The cost of new MSCs.
2. The cost of connecting BSCs to MSCs.
3. The cost of expanding the capacity of links connecting the MSCs.
4. A penalty cost for handovers that occur among BSCs that are connected to different MSCs.

The cost of a new MSC is a known (fixed) cost including the purchase price, physical installation in a building and a number of working hours for the installation. The cost of connecting a BSC to an MSC is zero if the BSC is currently connected to this MSC, otherwise the cost of moving the BSC to a new or existing MSC is estimated. The cost of expanding link capacities is a linear function of the number of new bandwidth units needed. Finally, a handover cost is introduced to keep BSC areas connected.
It is a fact, that the time that passes from the moment at which deployment of MSCs is resolved on until the equipment is actually in place and available for use is rather long (about a year). This means that at the time the decision has to be made the network provider does not have full knowledge about several important parameters such as the traffic matrix, the cost of expanding the capacity of links and so on. For this reason the network provider should put off the definitive decision on allocation of BSCs to MSCs as long as possible, allowing uncertainty to be at least partially revealed. This is the incentive for us to model the problem as a two-stage stochastic program, the first stage consisting of deployment of MSCs and the second stage consisting of allocation of BSCs to MSCs and routing of traffic in the resulting network.

7.1 A two-stage stochastic programming model

As previously discussed several parameters of the model are not known with certainty at the time the decision on deployment of MSCs has to be made. Thus we will think of these parameters as depending on the outcome of a random variable ξ defined on some probability space $(\Xi, \mathcal{F}, P)$. We will make the following assumption:

Assumption 1 *The random variable ξ has a discrete distribution with finite support $\Xi = \{\xi^1, \ldots, \xi^S\}$ and corresponding probabilities $P(\xi^1) = \pi^1, \ldots, P(\xi^S) = \pi^S$.*

Assumption 1 allows us to speak of the parameters in terms of scenarios, a scenario being a set of realized values of the parameters. $q(\xi^s)$ is the second stage prices, $h(\xi^s)$ is the second stage right hand side and $T(\xi^s)$ is the second stage effect of the first stage decision. For notational convenience we will refer to such a scenario simply by (q^s, h^s, T^s). In our case the only first stage parameters is the price vector c of deploying new MSCs, since it is possible (but not likely) to open all the possible new MSCs. The first stage (binary) variables are denoted $x = (x_1, \ldots, x_n)$, and the second stage variables are denoted y^s. Since our second stage problem is a MIP, y is split into y_1 and y_2 where y_1^s is a binary vector ($y_1^s \in \mathbb{B}^{m_1}$) and y_2^s is real vector ($y_2^s \in \mathbb{R}^{m_2}$). This gives the following model for minimizing the expected cost:

$$\begin{array}{ll} \min & cx + \sum_{s=1}^{S} \pi^s Q^s(x) \\ \text{s.t.} & \\ & x \in \mathbb{B}^n \end{array} \tag{22}$$

where the second stage value function $Q^s(x)$ is given by

$$\begin{array}{rcll} Q^s(x) & = & \min & q^s y^s \\ & & \text{s.t.} & \\ & & & W^s y^s = h^s - T^s x \\ & & & y_1^s \in \mathbb{B}^{m_1}, y_2^s \in \mathbb{R}^{m_2} \end{array} \tag{23}$$

We have not specified the details of the second stage program, but we will explain what kind of constraints it includes. The y_1^s binary variables connect BSCs to MSCs. One constraint set ensures that BSCs are only connected to open MSCs, and this has to be done in a way that the VLR capacities is not exceeded. The y_2^s real variables include both flow and excess flow variables. The flow variables represent the flow on a given edge, and the excess flow variable is used to price the installion of new link capacity. Finally, a constraint set is needed to measure the number of handovers. A handover is when two BSCs are connected to different MSCs.

We have omitted a commonly used set of constraints, namely the survivability constraints. These constraints ensures that alternative routes exists in case of edge failures, or that only a certain percentage of the traffic is lost. These constraints complicated our problem in a way that the model did not solve in a reasonable amount of time, but it is still a very important aspect of designing telecommunication networks.

7.2 Scenario decomposition

Even without the mentioned survivability constraints, the problem was difficult to solve. In this section we outline the procedure used, namely scenario decomposition (also called dual decomposition).

Scenario decomposition exploits the fact that the vast majority of variables and constraints in the stochastic program are scenario dependent. In fact the only thing tying the scenarios together are the first stage decisions on deployment of MSCs. For notational convenience we define the index-set of first stage decisions $V = \{1, \ldots, n\}$. If we use variable splitting on the first stage variables, defining a deployment of MSCs for each scenario $x^1, \ldots, x^S$, problem (22) becomes separable into independent scenario subproblems. The fact that the deployment of MSCs cannot be scenario dependent may now be represented by a *non-anticipativity constraint* stating the problem as:

$$\begin{array}{ll} \min & \sum_{s=1}^{S} \pi^s \left(cx^s + Q^s(x^s) \right) \\ \text{s.t.} & \\ & x^1 = x^2 = \ldots = x^S \\ & x^s \in \mathbb{B}^n \quad \forall s \in \{1, \ldots, S\} \end{array} \tag{24}$$

Relaxing the non-anticipativity constraint we obtain a problem which is completely separable into independent scenario subproblems. These subproblems are solved to obtain an optimal deployment of MSCs for each scenario. Next, non-anticipativity is reinforced by branching on components of these solutions which differ among scenarios. To be specific, we introduce a branching tree, initially consisting of only the root node corresponding to the original problem (22). In a given iteration we select a problem from the branching tree and solve the corresponding scenario subproblems obtaining scenario solutions $x^1, \ldots, x^S$. If MSC i is to be deployed in some scenario solutions and not in others we add two problems to the branching tree imposing for $s = 1, \ldots, S$ the constraints $x_i^s = 0$ and $x_i^s = 1$ respectively. Otherwise, if all scenario solutions are equal, we have a feasible solution of the original problem and may update the upper bound if appropriate. For a thourough description of such a procedure, including a Lagrangian relaxation of the non-anticipativity constraints, we refer to Carøe and Schultz [12].

Clearly, if the scenario subproblems are solved by means of some branch and bound procedure, some effort should be taken to put information from previous iterations in the above procedure to use. Thus, a node which is fathomed in a given subproblem in some iteration of the main procedure may be reconsidered in subsequent iterations since more variables are fixed as the main procedure progresses. In fact, for the problem instance considered in Section 7.3 the number of first stage variables was so small (less than 20) that an enumeration tree could be created a priori and used for all scenarios, thus precluding any reevalutions of nodes.

7.3 About the Sonofon problem

In this final section we will loosely describe our problem instance at Sonofon. Because of competitive conditions we cannot be too specific about the problem size and the input data. The problem has between 5 and 10 existing MSCs, less than 20 potential locations for new MSCs and less than 50 BSCs. The network interconnecting the MSCs is complete. The number of binary variables were reduced by dividing the area of interest into three regions and precluding from consideration certain allocations of BSCs to MSCs across regions. In the resulting formulation each scenario subproblem has 707 binary variables, 14598 continuous variables and 12045 constraints.

The cost of a new MSC is orders of magnitude higher than any other cost. The cost of connecting a BSC to an MSC was set to zero if the BSC is currently connected to this particular MSC, and otherwise the total cost of a movement was estimated. The cost of expanding link capacities is given by the total cost of installing new equipment. Finally, the handover costs were adjusted observing their effect on solutions, so as to create geographically connected BSC areas. The current demand for bandwidth and VLR-capacity was estimated from observations of traffic and the number of customers respectively.

The different cost terms are made scenario dependent by introducing stochastic fluctuations on the future prices. Likewise, future demand is calculated using the current observed demand scaled by different growth factors. We have used the following procedure to generate demand for VLR-capacity at BSC r under scenario s:

$$L_r^s = growth^s \cdot growth_r^s \cdot \text{Current demand}$$

where $growth^s$ is a parameter, sampled from a uniform distribution, which is used to reflect the average growth in the number of customers while $growth_r^s$ is a parameter, sampled from another uniform distribution, reflecting regional fluctuations from this average growth.

We have considered a four year time horizon with respect to customer demand even though the second stage decision is made after just one year. The reason for the four-year time horizon is to ensure a somewhat stable solution guaranteeing sufficient network capacity for three additional years beyond the completed deployment of new MSCs. This means that demand is in fact only partially revealed at the time the second-stage decisions are to be made, but since the additional information obtained at this point will provide an improved estimate of the true rate of growth in demand, the gain of postponing some decisions to the second stage is likely to be considerable.

The algorithm was implemented in C++ using procedures from the callable library from CPLEX 6.6. Considering 100 scenarios the solution times were about 3.5 hours CPU-time on a 700 MHz Linux PC. The solution suggested the deployment of one new MSC.

References

[1] M.J. Alves and J. Climaco. Using cutting planes in an interactive reference point approach for multiobjective integer linear programming problems. *European Journal of Operational Research*, 117:565–577, 1999.

[2] M.J. Alves and J. Climaco. An interactive reference point approach for multiobjective mixed-integer programming using branch-and-bound. *European Journal of Operational Research*, 124:478–494, 2000.

[3] J.A. Azevedo, M.E.O. Santos Costa, J.J.E.R. Silvestre Madeira, and E.Q.V. Martins. An algorithm for the ranking of shortest paths. *European Journal of Operational Research*, 69:97–106, 1993.

[4] R. Batta and S.S. Chiu. Optimal obnoxious paths on a network: Transportation of hazardous materials. *Operations Research*, 36:84–92, 1988.

[5] R. Benayoun, J. de Montgolfier, J. Tergny, and O. Laritchev. Linear programming with multiple objective functions: Step method (stem). *Mathematical Programming*, 1(3):366–375, 1971.

[6] O. Berman and Z. Drezner. A note on the location of an obnoxious facility on a network. *European Journal of Operational Research*, 120:215–217, 2000.

[7] O. Berman, Z. Drezner, and G.O. Wesolowsky. Routing and location on a network with hazardous threats. *Journal of the Operational Research Society*, 51:1093–1099, 2000.

[8] J. Brimberg and H. Juel. A bicriteria model for locating a semi-desirable facility in the plane. *European Journal of Operational Research*, 106:144–151, 1998.

[9] J. Brimberg and H. Juel. On locating a semi-desirable facility on the continuous plane. *International Transactions in Operational Research*, 5:59–66, 1998.

[10] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43:216–224, 1989.

[11] J.T. Buchanan. A naive approach for solving MCDM problems. *Journal of the Operational Research Society*, 48(2):202–206, 1997.

[12] C.C. Carøe and R. Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24:37–45, 1999.

[13] E. Carrizosa, E. Conde, and D. Romero-Morales. Location of a semiobnoxious facility. A biobjective approach. In 1996 Torremolinos, editor, *Advances in multiple objective and goal programming*, pages 338–346. Springer-Verlag, Berlin-Heidelberg, 1997.

[14] E. Carrizosa and F. Plastria. Location of semi-obnoxious facilities. *Studies in Locational Analysis*, 12:1–27, 1999.

[15] R.L. Church and R.S. Garfinkel. Locating an obnoxious facility on a network. *Transportation Science*, 12:107–118, 1978.

[16] J.C.N. Climaco and E.Q.V. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.

[17] J.L. Cohon. *Multiobjective Programming and Planning.* Academic Press, 1978.

[18] H.W. Corley and I.D. Moon. Shortest paths in networks with vector weights. *Journal of Optimization Theory and Applications*, 46:79–86, 1985.

[19] 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.

[20] H.G. Daellenbach and C.A. De Kluyver. Note on multiple objective dynamic programming. *Journal of the Operational Research Society*, 31:591–594, 1980.

[21] Z. Drezner and G.O. Wesolowsky. Location of multiple obnoxious facilities. *Transportation Sci.*, 19:193–202, 1985.

[22] Z. Drezner and G.O. Wesolowsky. The weber problem on the plane with some negative weights. *INFOR*, 29:87–99, 1991.

[23] M. Ehrgott. On matroids with multiple objectives. *Optimization*, 38(1):73–84, 1996.

[24] M. Ehrgott. *Multicriteria Optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems.* Springer, 2000.

[25] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multicriteria combinatorial optimization. *OR Spektrum*, 22:425–460, 2000.

[26] M. Ehrgott, S. Nickel, and H.W. Hamacher. Geometric methods to solve max-ordering location problems. *Discrete Applied Mathematics*, 93:3–20, 1999.

[27] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[28] E. Erkut and S. Neuman. Analytical models for locating undesirable facilities. *European Journal of Operational Research*, 40:275–291, 1989.

[29] E. Erkut and V. Verter. Modeling of transport risk for hazardous materials. *Operations Research*, 46:625–642, 1998.

[30] J. Fernández, P. Fernández, and B. Pelegrín. A continuous location model for siting a non-noxious undesirable facility within a geographical region. *European Journal of Operational Research*, 121:259–274, 2000.

[31] R.L. Francis, L.F. McGinnis, and J.A. White. *Facility layout and location: An analytical approach.* Prentice Hall, New Jersey, 1992.

[32] H.N. Gabow. Two algorithms for generating weighted spanning trees in order. *SIAM Journal of Computing*, 6(1):139–150, 1977.

[33] M.R. Garey and D.S. Johnson. *Computers and Intractability. A guide to the Theory of $\mathcal{NP}$-Completeness.* W.H.Freeman, San Francisco, 1979.

[34] A.M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22:618–630, 1968.

[35] H.W. Hamacher. A note on K best network flows. *Annals of Operations Research*, 57:65–72, 1995. Special Volume "Industrial Systems".

[36] H.W. Hamacher, M. Labbe, and S. Nickel. Multicriteria network location problems with sum objectives. *Networks*, 33:79–92, 1999.

[37] H.W. Hamacher and S. Nickel. Combinatorial algorithms for some 1-facility median problems in the plane. *European Journal of Operational Research*, 79:340–351, 1994.

[38] H.W. Hamacher and S. Nickel. Multicriteria planar location problems. *European Journal of Operational Research*, 94:66–86, 1996.

[39] H.W. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4:123–143, 1985.

[40] P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, editors, *Multiple criteria decision making: theory and applications*, Lecture Notes in Economics and Mathematical Systems 177, pages 109–127. Springer-Verlag, Heidelberg, 1980.

[41] P. Hansen, M. Labbè, D. Peeters, and J.F. Thisse. Single facility location on networks. *Annals of Discrete Mathematics*, 31:113–146, 1987.

[42] P. Hansen, M. Labbè, and J.F. Thisse. From the median to the generalized center. *RAIRO Rech. Opér.*, 25:73–86, 1991.

[43] P. Hansen, D. Peeters, D. Richard, and J.F. Thisse. The minisum and minimax location problems revisited. *Operations Research*, 33:1251–1265, 1985.

[44] P. Hansen, D. Peeters, and J.F. Thisse. On the location of an obnoxious facility. *Sistemi Urbani*, 3:299–317, 1981.

[45] J. Hershberger. Finding the upper envelope of n line segments in $O(n\ log\ n)$ time. *Info Process Lett*, 33:169–174, 1989.

[46] N. Katoh, T. Ibaraki, and H. Mine. An algorithm for finding k minimum spanning trees. *SIAM Journal of Computing*, 10(2):247–255, 1981.

[47] M. Labbé. Location of an obnoxious facility on a network: A voting approach. *Networks*, 20:197–207, 1990.

[48] E.L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1972.

[49] R.F. Love, J.G. Morris, and G.O. Wesolowsky. *Facilities Location : Models & Methods*. North-Holland, New York, 1988.

[50] E.Q.V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984.

[51] E.Q.V. Martins, M.M.B. Pascoal, and J.L.E. Dos Santos. A new improvement for a k shortest paths algorithm. Technical report, Universidade de Coimbra, 2000.

[52] E.Q.V. Martins, M.M.B. Pascoal, and J.L.E. Santos. Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10:247–261, 1999.

[53] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput*, 12:759–776, 1983.

[54] E. Minieka. Anticenters and antimedians of a network. *Networks*, 13:359–364, 1983.

[55] J. Mote, I. Murthy, and D.L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53:81–92, 1991.

[56] I. Murthy and S. Her. Solving min-max shortest path problems on a network. *Naval Research Logistics*, 39:669–683, 1992.

[57] S. Nickel and J. Puerto. A unified approach to network location problems. *Networks*, 34:283–290, 1999.

[58] P.H. Peeters. Some new algorithms for location problems on networks. *European Journal of Operational Research*, 104:299–309, 1998.

[59] F. Plastria. The generalized big square small square method for the planar single-facility location. *European Journal of Operational Research*, 62:163–174, 1992.

[60] F. Plastria and E. Carrizosa. Undesirable facility location with minimal covering objectives. *European Journal of Operational Research*, 119:158–180, 1999.

[61] J. Puerto and F.R. Fernandez. Multi-criteria minisum facility location problems. *J. Multi-Crit. Decis. Anal.*, 8:268–280, 1999.

[62] K. Rana and R.G. Vickson. A model and solution algorithm for optimal routing of a time-chartered containership. *Transportation Science*, 22:83–96, 1988.

[63] L.M. Rasmussen. Zero-one programming with multiple criteria. *European Journal of Operational Research*, 26:83–95, 1986.

[64] D. Romero-Morales, E. Carrizosa, and E. Conde. Semi-obnoxious location models: A global optimization approach. *European Journal of Operational Research*, 102:295–301, 1997.

[65] A.J.V. Skriver. A classification of bicriteria shortest path (BSP) algorithms. *Asia-Pacific Journal of Operations Research*, 17:199–212, 2000.

[66] A.J.V. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest path problems. *Computers and Operations Research*, 27:507–524, 2000.

[67] R.S. Solanki, P.A. Appino, and J.L. Cohon. Approximating the noninferior set in multiobjective linear programming problems. *European Journal of Operational Research*, 68:356–373, 1993.

[68] R.E. Steuer. *Multiple criteria optimization: Theory, Computation, and Application*. Wiley, New York, 1986.

[69] R.E. Steuer and E.U. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26:326–344, 1983.

[70] C.S. Sung and C.M. Joo. Locating an obnoxious facility on a euclidian network to minimize neighborhood damage. *Networks*, 24:1–9, 1994.

[71] K. Thulasiraman and M.N.S. Swamy. *Graphs: Theory and Algorithms*. Wiley, New York, 1992.

[72] C.T. Tung and K.L. Chew. A bicriterion pareto-optimal path algorithm. *Asia-Pacific Journal of Operations Research*, 5:166–172, 1988.

[73] E.L. Ulungu and J. Teghem. The multi-objective shortest path problem: A survey. In Glückaufova Cerny and Loula, editors, *Proceedings of the International Workshop on Multicriteria Decision Making: Methods - Algorithms - Applications at Liblice*, pages 176–188. Czechoslovakia, 1991.

[74] E.L. Ulungu and J. Teghem. The two-phases method: An efficient procedure to solve biobjective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20:149–165, 1995.

[75] M. Visée, J. Teghem, M. Pirlot, and E.L. Ulungu. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization*, 12:139–155, 1998.

[76] A. Warburton. Aproximation of Pareto optima in multiple-objective shortest-path problems. *Operations Research*, 35(1):70 –79, 1987.

[77] S. Zionts. A survey of multiple criteria integer programming methods. *Annals of Discrete Mathematics*, 5:389–398, 1979.

A Classification of Bicriterion Shortest Path (BSP) Algorithms

ANDERS J.V. SKRIVER
Department of Operations Research
University of Aarhus, building 530
Ny Munkegade
DK - 8000 Århus C
Denmark

August 16, 2001

Abstract

This is a survey paper with references to relevant papers in the field of the Bicriterion Shortest Path (BSP) problem. It classifies the algorithms by their structure to argue theoretically how they will perform, and at least one algorithm from each class is discussed in more detail.

Keywords: MOLP, MCDM, MOIP, Bicriterion, Shortest Path.

1 Introduction

The Bicriterion Shortest Path (BSP) problem is one of the simplest problems in multicriterion linear integer analysis, but nevertheless also one of great importance in many applications. For example it is of interest to model transportation problems with more than one objective, e.g. cost and travel time. Also, the BSP problem often occurs as a subproblem in other problems, for example in scheduling problems. This paper is an overview of the existing literature in the field. For previous survey papers see Zionts [19], Rasmussen [15] and Ulungu and Teghem [18]. The first two references survey the general multicriteria integer programming problem for which the BSP is a special case, and both papers are now quite old. The last reference surveys many of the papers also discussed in this paper. The main contribution of this paper is a classification of the different solution methods, and a ranking of the procedures based on the algorithmic structure.

Let us describe the problem. We have a *strongly connected directed network* or a *digraph* $G = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{1, \ldots, n\}$ is the set of nodes and $\mathcal{A} = \{(i,j), (k,l), \ldots, (p,q)\}$ is a finite set of directed edges (arcs) joining nodes in $\mathcal{N}$. Assume we have $|\mathcal{A}| = m$ edges. Each edge $(i,j) \in \mathcal{A}$ carries two attributes denoted by (c_{ij}, t_{ij}). For simplicity assume that c_{ij} is the cost using edge (i,j) and t_{ij} is the travel time. The objective is to find a "shortest" path from a particular node, the source node $s \in \mathcal{N}$, to another particular node, the terminal node $t \in \mathcal{N}$. Traditionally, the BSP problem is formulated as follows:

$$\begin{aligned}
\min \quad & c(x) = \textstyle\sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \\
\min \quad & t(x) = \textstyle\sum_{(i,j)\in\mathcal{A}} t_{ij}x_{ij} \\
\text{s.t.} \quad & \\
& \textstyle\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} \quad - \quad \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ 0 & \text{if } i \neq s,t \\ -1 & \text{if } i = t \end{cases} \\
& x_{ij} \in \{0,1\}, \;\; \forall \;\; (i,j) \in \mathcal{A}
\end{aligned} \tag{1}$$

The constraints in (1) yield a directed path from source node s to terminal node t and the two objectives are to find the minimum cost $s-t$ path and the minimum travel time $s-t$ path, respectively.

It is highly unlikely to find a directed path from node s to node t which achieves both the minimum total cost and the minimum total travel time. We therefore have to settle with something less, namely finding the set of efficient paths (see Definition 1) from node s to node t.

The problem is known to be $\mathcal{NP}$-complete by transformation from a 0-1 knapsack problem, Garey and Johnson [8], and Hansen [10] give an example with exponentially many distinct efficient paths (intractable). Next, we define efficient points (paths) and nondominated criterion vectors. Let $z(x) = (c(x), t(x))$.

Definition 1 *A point x that satisfies the constraints of (1) is **efficient** iff there does not exist a point $\overline{x}$ that satisfies the constraints (1) such that $z(\overline{x}) \leq z(x)$ with at least one strict inequality. Otherwise x is **inefficient**.*

Please note that efficient points are the same as Pareto optimal points. Efficient points are defined in decision space. There is a natural counterpart in criterion space, where the criterion space $\mathcal{Z}$ is defined as $\mathcal{Z} = \{z(x) \in R^2 | x \text{ satisfies the constraints in (1)}\}$. So the criterion vectors correspond to the image of a linear mapping of all the feasible solutions to (1).

Definition 2 *$z(x) \in \mathcal{Z}$ is a **nondominated** criterion vector iff x is an efficient solution. Otherwise $z(x)$ is a **dominated** criterion vector.*

We define the combined objective function $W(x, \lambda)$ as follows:

$$W(x,\lambda) = \lambda c(x) + (1-\lambda)t(x) \quad 0 < \lambda < 1 \tag{2}$$

The function $W(x, \lambda)$ is a convex combination, or **weighted sum**, of the two objective functions. Optimizing this function with different λ's will give the so-called **supported** nondominated solutions and is therefore often referred to as the **weighting method**. Since unsupported nondominated criterion vectors are dominated by a convex combination of supported nondominated criterion vectors, unsupported nondominated vectors cannot be found by the weighting method. This is illustrated in Figure 1. The solution(s) x in decision space corresponding to a supported criterion vector can be referred to as a supported solution.

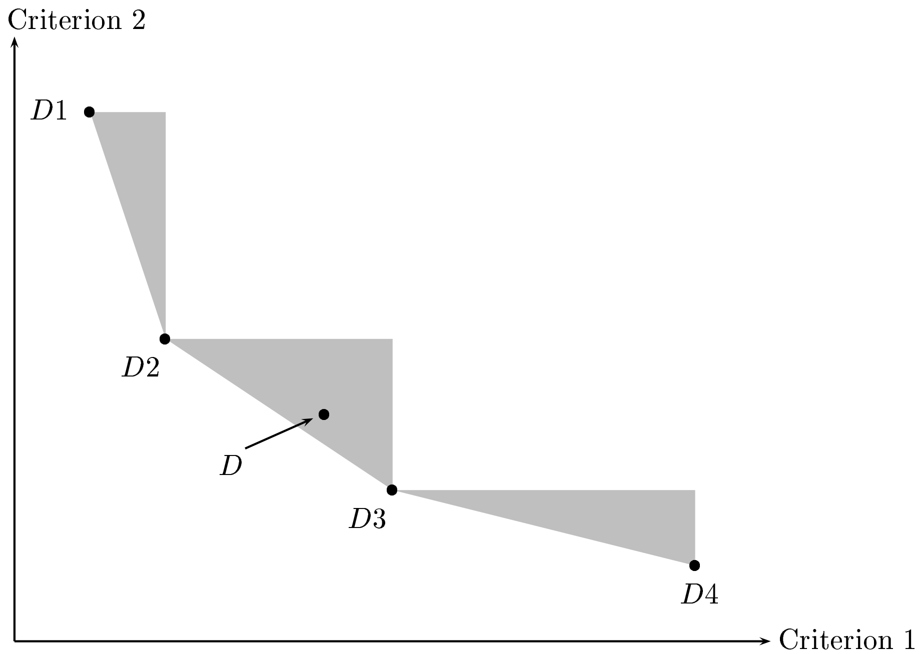

Figure 1: D is an unsupported, nondominated criterion vector.

We know from basic Mathematical Programming (e.g. [1]), that the solutions to the linear relaxation of (1) with objective function (2) are integer valued, because the unimodularity property holds. The points $D1, D2, D3$ and $D4$ in Figure 1 illustrate solutions to (2), with different values of λ. The shaded areas are nondominated regions defined by those four points. The point D inside one of the shaded areas, is therefore nondominated in (1), but it is dominated in (2). The conclusion is that we cannot, in general, find all the efficient solutions as supported solutions. We have to search in between the supported paths as well.

The rest of the paper is organized as follows. In Section 2 we describe some of the contributions to solve the problem. Many of the algorithms are presented. We conclude the paper in Section 3.

2 Description of algorithms

Our objective is to find the complete set of efficient solutions, or the complete set of nondominated criterion vectors. Only algorithms that fulfill this goal are included.

There are generally two main approaches, a path/tree approach and a node labeling approach, see Figure 2. Each of the two main approaches are again divided in two. The path/tree approach splits into the K'th shortest path approach and the Two Phases method. The node labeling approach splits into a Label Setting and a Label Correcting approach.

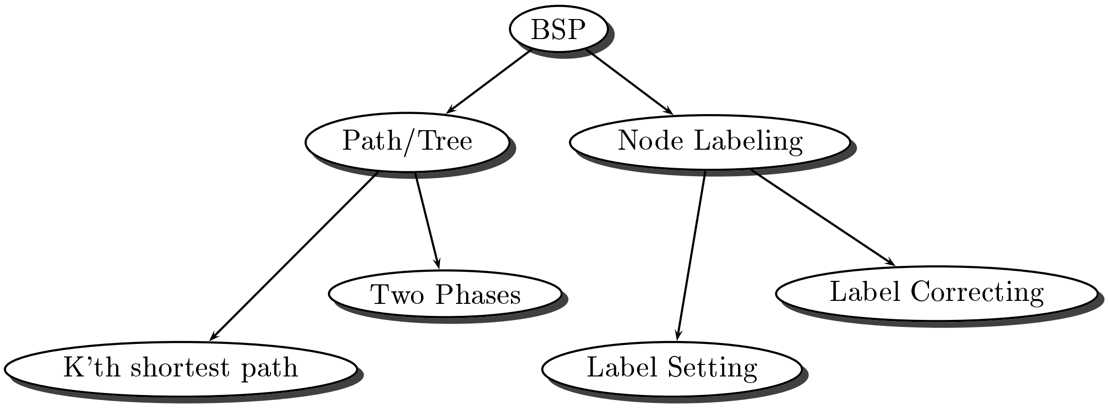

Figure 2: Classification of BSP algorithms.

In a path approach we examine different path vectors, and try to find the efficient ones. Similarly, we investigate the m dimensional incidence vectors that characterize the different spanning trees in a tree approach. Since there are usually many edges compared to the number of nodes and there may be exponentially many spanning trees, a labeling approach that compares values in the two-dimensional criterion space at each node may be advantageous. In a Label Setting approach one label is made permanent in each iteration and in a Label Correcting approach all labels are changeable until the stop criterion is fulfilled.

In Table 1 I list the references that fall in the four categories. The number of references applying a labeling approach indicates that this is the most successful approach. In a joint paper with K.A. Andersen, [16], we describe in detail, why the node labeling approach is to be preferred for the BSP problem.

To clarify the similarities and differences between the different labeling algorithms, I outline a generic labeling algorithm. Each node has a set of labels associated with it. Denote the label-set at node i by $L(i)$. This set contains labels of the form (c, t)

Table 1: Classification of references.

K'th shortest path	[4]
Two Phases	[14]
Label Setting	[10], [13] algorithm 1, [17]
Label Correcting	[2], [5], [6], [16]

sorted by increasing c-values (and decreasing t-values). At node-level we assume that dominated labels are deleted. Let $L = \cup_{i=1}^{n} L(i)$ be the set of all labels, again sorted by increasing c (but not necessarily decreasing t). Dominated labels are not deleted, because they belong to different nodes. The merge operation on the sets A and B is defined as $\mathit{Merge}(A, B) = (A \cup B) \setminus \{z \in A \cup B \mid \exists x \in A \cup B : x \leq z\}$. This means that after the sets are joined all dominated labels are deleted. Note that the set A could be just one label.

Generic labeling algorithm:

1. Initialization: Label node s, $L(s) = \{(0, 0)\}$
2. Select a node i by some rule
3. Generation of new labels using node i
4. Stop or go to Step 2

The labeling algorithms differ in node/label selection rule, label generation and stopping criterion, but they all have the structure of the generic algorithm. In the following sections, the references are discussed in more detail, and when possible related to the generic algorithm.

2.1 Climaco and Martins [4]

Some basic theory in the field is provided by Climaco and Martins [4] along with an algorithm. They use an upper bound on the cost criterion as a stopping criterion. If we minimize the time criterion we get the fastest path. Choose among these fastest paths the cheapest one. This cost value is denoted $\hat{c}$. Observe that $\hat{c}$ is the best value of the cost objective, given the time objective is at its global minimum. Objective vectors with c values higher than $\hat{c}$ are therefore obviously dominated.

2.1.1 Climaco and Martins' algorithm

Climaco and Martins' algorithm uses an ordered search starting by minimizing the cost criterion and searching for the best value of the time criterion. Then the cost criterion is gradually relaxed, each time finding the best path with respect to the time criterion. This continues until the value of the cost criterion exceeds $\hat{c}$.

Algorithm 2.1.1:

1. Initialize:

   Compute $\hat{c}$ as upper bound on c

   Find p_1 as the cheapest path

   Set $\mathcal{S} = \{p_1\}$ and $K = 2$

2. Compute the K'th cheapest path p_K
3. If $c(p_K) > \hat{c}$ stop, $\mathcal{S}$ is the set of efficient paths
4. If $t(p_K) \geq t(p_{K-1})$ then set $K = K + 1$ and go to 2
5. If $t(p_K) < t(p_{K-1})$ then $\mathcal{S} = \mathcal{S} \cup \{p_K\}$, set $K = K + 1$ and go to Step 2

To keep the algorithm in a compact form, we assume that $c(p_{K+1}) > c(p_K)$ $\forall K \in 1, 2, 3, \ldots$. This is not a restrictive assumption.

Due to the K'th shortest path routine included, it has little hope for being fast. The K'th shortest path problem is intractable for general K, which in this case means that we may have to enumerate all solutions. It is, however, polynomial for fixed K, but in our case K is unknown and expected to be very large. In Climaco and Martins [4] they refer to Lawler [12] for the K'th shortest path algorithm. Alternative procedures are found in [3], and more recently [7]. Lawler's algorithm works by forbidding the $K - 1$ shortest paths in order to find the K'th shortest.

2.2 Mote, Murthy and Olson [14]

A paper by Mote, Murthy and Olson [14] uses a Two-Phases method. Instead of the K'th shortest path approach of Section 2.1, they use the unimodularity property of the network constraints in (1) to find the Pareto optimal supported paths (Phase I). In Phase II the unsupported solutions are found by a Label Correcting algorithm. They call their approach the Parametric Approach, because of their use of the weighting method in Phase I (λ is a weighting parameter).

Two important results are used. The first is the use of Geoffrion's result, [9], to determine the weights of the convex combination of the two objectives in order to move from one supported solution to the next supported solution. Geoffrion's result ensures that no supported solutions are in between. This movement between solutions with increasing cost and decreasing time is similar to the idea of Climaco and Martins' algorithm. The second important result is that a supported path uses only supported sub-paths, and that an unsupported $s-t$ path uses a supported path from s to some node j, and then some unsupported $j-t$ path. This result is the basis of Phase II.

In Phase I the authors find the efficient supported paths from s to all other nodes in the network as minimal spanning tree solutions to the linear relaxation of the BP-problem (3) explained below. Note that the spanning trees are rooted at s.

The BP problem is formulated as follows, with the same objective functions as in (1):

$$\begin{array}{ll} \min & c(x) \\ \min & t(x) \\ \text{s.t.} & \\ & \sum_{(i,j)\in\mathcal{A}} x_{ij} \quad - \quad \sum_{(j,i)\in\mathcal{A}} x_{ji} = \begin{cases} n-1 & \text{if } i = s \\ -1 & \text{if } i \neq s \end{cases} \\ & x_{ij} \in \ \{0,1,2,\ldots,n-1\} \end{array} \tag{3}$$

Please note that the x_{ij} is now integer values and not just 0 and 1. If, say $x_{ij} = 3$ in a solution, this means that edge (i,j) is used in three different paths. One of these is for sure the path from s to j. This is illustrated in Figure 3 for a network with 5 nodes.

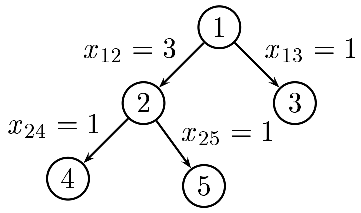

Figure 3: Example of a spanning tree solution for the constraints in (3).

Because of this spanning tree phase, I have classified the algorithm as a tree algorithm. This approach (Phase I) with spanning trees is similar to algorithm 2 presented in Martins [13]. Martins' algorithm 2 is not included in this paper, but algorithm 1 of the paper is presented in Section 2.3. Different procedures to obtain the supported nondominated solutions could be used, e.g. repeated use of Dijkstra's shortest path algorithm.

The principle of the Phase II algorithm is to add arbitrary edges to the supported $s - j$ paths in order to create alternative unsupported paths. The objective values are then evaluated, candidates for efficient paths are labeled, and dominated solutions are deleted. Phase II is very similar to Brumbaugh-Smith and Shier [2] presented in Section 2.6, except for the label generation being done by expanding on only one label at a time.

2.2.1 Mote, Murthy and Olson's algorithm

The set $L(i)$ contains the supported nondominated labels at node i, and the set $T(i)$ contains the unsupported nondominated labels. We say a node is being **scanned** if it is used to generate new labels.

Algorithm 2.2.1:

Phase I:

1. Initialize $L(i)$ and $T(i)$ as empty sets
2. Find all the supported nondominated labels $L(i)$, by use of the weighting method
3. Let $T(i) = L(i) \quad \forall i \in \mathcal{N}$. Go to Phase II

Phase II:

1. If all labels in $T(i)$ for all $i \in \mathcal{N}$ are scanned, go to Step 3. Else select an unscanned label $(c, t)(i) \in T(i)$
2. (a) Compute label $(c, t)(j) = (c, t)(i) + (c_{ij}, t_{ij}) \; \forall (i, j) \in \mathcal{A}$
   (b) For each $(c, t)(j)$ found in (a), Merge $(c, t)(j)$ with $T(j)$
   (c) Go to Step 1
3. Report all efficient solutions from s to t found in $L(t)$ and $T(t)$

One may think that this algorithm is fast for finding supported efficient paths, due to unimodularity, but I have found that the Label Correcting algorithm is much more effective, Skriver and Andersen [16]. In fact Huarng, Pulat and Shih [11] (Section 2.9) have found the Label Correcting algorithm by Brumbaugh-Smith and Shier [2] (Section 2.6) to be significantly faster, even compared with Phase I only.

2.3 Martins [13]

This paper presents a Label Setting algorithm. It is a multicriteria generalization of Hansen's bicriteria algorithm, [10], briefly mentioned in Section 2.4. The algorithm can be seen as a generalization of Dijkstra's shortest path algorithm to multiple criterias. It is assumed that all edge-coefficients (c_{ij}, t_{ij}) are non-negative.

In each iteration in the algorithm we choose the lexicographically smallest label in the set L of all labels. This is where the assumption of non-negative weights is needed. When finding the lexicographically smallest vector, we first look at the first coordinates. If only one vector has the smallest first coordinate, this is the lexicographically smallest vector. If more vectors have equal first coordinates, we look at the second coordinates and so on.

2.3.1 Martins' algorithm

The algorithm makes a set of labels at each node. The labels are all put in a set L, and at each iteration one label is removed as permanently labeled.

From all the permanent labels at the terminal node t, the DM can choose the label l with the cost/time combination that he/she prefers. Then this particular path can be found by backtracking. Next I will explain Martins' algorithm by describing the steps of the generic algorithm.

Step 2: The selection rule is to choose the lexicographically smallest label from all the labels and remove this label from L as permanently labeled. Assume this label is $(c,t)(i) \in L(i)$.

Step 3: When generating new labels, the label $(c,t)(i)$ above is used. We generate new labels for all the nodes that can be reached from node i. Assume edge $(i,j) \in \mathcal{A}$. We then Merge the new label $(c,t)(i) + (c_{ij}, t_{ij})$ with $L(j)$.

Step 4: We stop when $L = \emptyset$.

It can be seen that one label is labeled permanently in each iteration. By choosing the lexicographically smallest, we ensure that this is always a nondominated label. In this perspective it can be seen that the algorithm relies on the fact that a nondominated path uses only nondominated sub-paths.

The complexity of the algorithm is hidden in Step 3. Here we generate new labels, and for each new label also check for domination. Both operations are time consuming, because they have to be done a large number of times.

In the paper by Martins [13] a second algorithm is presented. This algorithm is similar in structure to the first phase of the Two Phases method described in Section 2.2, in the way that it alternates between spanning trees by looking at the reduced costs. This algorithm is not presented here.

2.4 Hansen [10]

Hansen was one of the first to propose an algorithm for the BSP problem (back in 1980). His algorithm is explained in detail in Section 2.3, because Martins' algorithm is a generalization of Hansen's algorithm. The original paper by Hansen contains 10 different BSP problems and solution procedures for these different problems. It also contains an example that explains the complexity of the problem. His algorithm has been used for comparison with later algorithms both in Mote, Murthy and Olson [14] and Huarng, Pulat and Shih [11].

2.5 Tung and Chew [17]

The paper by Tung and Chew [17], suggests a forward labeling algorithm. The algorithm starts in an optimistic manner by moving in the direction of the minimum sum of the two criteria, and then labels the possible next steps. When visited, the labels are updated, and finally labeled permanently. The algorithm has not been implemented for testing. The structure of the algorithm is a Label Setting structure.

2.6 Brumbaugh-Smith and Shier [2]

The authors of this paper present a Label Correcting algorithm. They use some effort on implementation issues, and find that the CPU-times depend heavily on the way the different label-sets are scanned (Step 2) and deleted (Step 3). The worst principle LIFO (Last In First Out) is more than a factor 10 slower than the fastest principle, namely FIFO (First In First Out).

The most encouraging result is the fact that the computational effort grows linearly with the number of edges in the networks and sub-linearly with the average number v of labels (at the nodes). The total number of labels are therefore the number of elements in L. The following statistical model is found to have good fit:

$$T = \alpha m^{\beta} v^{\gamma} \tag{4}$$

where T is CPU-time, m is the number of edges, and v is the average number of labels per node. α is just a constant depending on the CPU-time units. The parameter β is found to be just less than one, and γ is found to be just less than one half. It is, however, hard to believe that this result is true for large networks.

The last result is on correlation of the objectives, and states that the number of efficient paths increase, when the objectives get more negatively correlated. This is to be expected. The authors find the increase in the number of efficient paths as a function of correlation to be rapidly increasing when the correlation is from -0.6 to -1. It can be noted that the number of efficient paths is equal to the number of nodes (one efficient path from the source to each of the other nodes including itself), when the correlation is 1.

2.6.1 Brumbaugh-Smith and Shier's algorithm

This algorithm is also outlined by describing the different steps of the generic algorithm.

Step 1: We form a list **Labeled** of nodes to be scanned, initially the node s. The selection rule is optional, but we investigate a label-set $L(i)$ and not just a particular label $(c,t)(i) \in L(i)$. In the paper the FIFO principle is suggested. Assume we select node $i \in$ Labeled.

Step 2: When generating new labels, all labels in $L(i)$ are used. We generate new labels for all the nodes that can be reached from node i. Assume edge $(i,j) \in \mathcal{A}$. We then Merge the new labels $L(i) + (c_{ij}, t_{ij})$ with $L(j)$. If $L(j)$ changes, add node j to the list Labeled.

Step 3: We stop when Labeled $= \emptyset$.

The label-set $L(t)$ contains all the nondominated labels for the efficient paths from s to t. The time-consuming part of the algorithm is the Merge operation (in Step 3), even though this is in linear time in the size of the two sets. Also note that in this algorithmic structure, a node can return to the set Labeled a large number of times. In this algorithm we have a choice of how to choose the nodes from the set Labeled. Different rules (policies) for doing this is discussed in detail in Brumbaugh-Smith and Shier [2].

2.7 Corley and Moon [5]

I have chosen to present this Label Correcting algorithm too, because it has a different label generation procedure than Brumbaugh-Smith and Shier [2] described in Section 2.6. This algorithm is in fact a generalization of Ford and Bellmann's shortest path algorithm (see [7], p. 88-89). The paper also presents a sub-algorithm for the Merge operation.

This algorithm can detect negative cycles, and negative weights are therefore allowed.

2.7.1 Corley and Moon's algorithm

Let $L_k(i)$ be the label-set at node i after k iterations. In each iteration k we try to improve $L_{k-1}(i)$ at each node i by using an intermediate node j. The algorithm terminates when this is no longer possible (or when a negative cycle is detected). The iteration counter k is the maximum number of edges used in a path from node 1 to node i after iteration k.

Step 2: Select each node in turn by the node number, $1, 2, \ldots, n$. Assume we are looking at node i.

Step 3: We generate new labels for node i by expanding the label-sets of all nodes j, where edge $(j, i) \in \mathcal{A}$. Assume we are currently expanding from node j. We then Merge the new labels $L(j) + (c_{ji}, t_{ji})$ with $L(i)$. When generating new labels, all labels in $L(j)$ are used.

Step 4: We stop when $k = n - 1$ or $L_k(i) = L_{k-1}(i) \quad \forall i \in \mathcal{N}$.

When the algorithm stops, the label $L(t)$ at the terminal node t contains the nondominated values from node 1 to node t. All efficient paths will consist of k or fewer edges, in iteration k. If the algorithm terminates with $k = n - 1$ there exists a negative cycle.

When we compare the two Label Correcting algorithms we see that they are very similar. The main difference is that in Brumbaugh-Smith and Shier [2] we have a choice of selection rule for the set Labeled, and only nodes with changes in their label-sets are re-examined. In Corley and Moon [5] we each time add labels with one more edge than in the iteration before.

If we compare the Label Setting and the two Label Correcting algorithms, the main difference is that we only expand on one label, namely the one recently made permanent, when we form new labels in the Label Setting algorithm. In the Label Correcting algorithm we expand on the set $L(i)$ at a particular node i.

2.8 Daellenbach and DeKluyver [6]

This paper presents an algorithm similar to Brumbaugh-Smith and Shier's algorithm described in Section 2.6, but it is formulated in the context of dynamic programming. The different steps of the algorithm are very generally defined, but it is essentially the same structure. The difference being that **no cycles are allowed** in this dynamic programming context. Edges are only allowed to point to nodes with higher numbers. This assumption is not explicit in the paper, but it is essential for the algorithm.

Instead of using a set Labeled for the changing labels as in Brumbaugh-Smith and

Shier [2], they move from one node to the next. Therefore the assumption of no cycles is needed, and this is a very restrictive assumption.

2.9 Huarng, Pulat and Shih [11]

This paper is a comparison of some of the existing algorithms. Some of the algorithms appear differently from the algorithms of the original papers. Their K'th shortest path implementation of Climaco and Martins [4] does not seem to find all efficient paths, as it is constructed to do, and the Two Phases method of Mote, Murthy and Olson [14] has a different Phase I implementation.

Despite this criticism, their computational results suggest that the Label Correcting approach ([2] implemented) is the fastest approach. This is in fact the approach we have improved in [16]. They also find that the Label Setting approach is far better than both the K'th shortest path approach and the Two Phases method.

2.10 Skriver and Andersen [16]

By imposing some preprocessing conditions to the Label Correcting algorithm by Brumbaugh-Smith and Shier [2] (Section 2.6) in each iteration, we have saved more than 50 % in CPU-time on some of our random networks. How much CPU-time is saved depends on the network structure. Our algorithm is the fastest algorithm for the BSP problem at the moment. This suggests that the Label Correcting approach is the best known for the BSP.

The paper also contains a discussion on the structure of random networks. The structure of the random networks has great impact on the computational results, and we have made a program that generates what we believe is realistic random networks for testing.

3 Concluding remarks

With the number of algorithms implemented, and the computational results found, many real life problems can now be modeled with more than one objective. This may lead to a more realistic representation of the problem. Most of the algorithms discussed can be easily modified to handle more than two objectives, making even more sophisticated models applicable. This is in fact the case for all the labeling algorithms.

We have seen that there are generally two types of algorithms, path/tree and labeling. I argue that the path/tree approach has been the least successful of the two. The labeling algorithms, and the Label Correcting approach in particular, performs much better. For the Label Correcting algorithm, the order in which the labels are selected and expanded, can result in significant differences in the running times.

Finally, one may think that finding the supported nondominated solutions by combining the weighting method and some shortest path algorithm can be done relatively fast. This does not seem to be the case! The repeated use of, e.g. Dijkstra's shortest path algorithm, seems to be a slower approach, than applying the Label Correcting approach, finding all nondominated solutions at once.

Acknowledgments

The author would like to thank Kim Allan Andersen, Matthias Ehrgott and Philip Melchiors for constructive criticism. I would also like to thank the two anonymous reviewers for very constructive referee reports.

References

[1] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear Programming and Network Flows.* Wiley, New York, 2nd edition, 1990.

[2] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43:216–224, 1989.

[3] N. Christofides. *Graph Theory: An Algorithmic Approach.* Academic Press, London, 1975.

[4] J.C.N. Climaco and E.Q.V. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.

[5] H.W. Corley and I.D. Moon. Shortest paths in networks with vector weights. *Journal of Optimization Theory and Applications*, 46:79–86, 1985.

[6] H.G. Daellenbach and C.A. De Kluyver. Note on multiple objective dynamic programming. *Journal of the Operational Research Society*, 31:591–594, 1980.

[7] J.R. Evans and E. Minieka. *Optimization Algorithms for Networks and Graphs.* Dekker, New York, 2nd edition, 1992.

[8] M.R. Garey and D.S. Johnson. *Computers and Intractability. A guide to the Theory of $\mathcal{NP}$-Completeness.* W.H.Freeman, San Francisco, 1979.

[9] A.M. Geoffrion. Solving bicriterion mathematical programs. *Operations Research*, 15:39–54, 1967.

[10] P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, editors, *Multiple criteria decision making: theory and applications*, Lecture Notes in Economics and Mathematical Systems 177, pages 109–127. Springer-Verlag, Heidelberg, 1980.

[11] F. Huarng, P.S. Pulat, and L. Shih. A computational comparison of some bicriterion shortest path algorithms. *J. Chinese Inst. Indust. Engrs.*, 13:121–125, 1996.

[12] E.L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1972.

[13] E.Q.V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984.

[14] J. Mote, I. Murthy, and D.L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53:81–92, 1991.

[15] L.M. Rasmussen. Zero-one programming with multiple criteria. *European Journal of Operational Research*, 26:83–95, 1986.

[16] A.J.V. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest path problems. *Computers and Operations Research*, 27:507–524, 2000.

[17] C.T. Tung and K.L. Chew. A bicriterion pareto-optimal path algorithm. *Asia-Pacific Journal of Operations Research*, 5:166–172, 1988.

[18] E.L. Ulungu and J. Teghem. The multi-objective shortest path problem: A survey. In Glückaufova Cerny and Loula, editors, *Proceedings of the International Workshop on Multicriteria Decision Making: Methods - Algorithms - Applications at Liblice*, pages 176–188. Czechoslovakia, 1991.

[19] S. Zionts. A survey of multiple criteria integer programming methods. *Annals of Discrete Mathematics*, 5:389–398, 1979.

A label correcting approach for solving bicriterion shortest path problems

Anders J. V. Skriver *
Kim Allan Andersen

Department of Operations Research
University of Aarhus, building 530
Ny Munkegade
DK - 8000 Århus C
Denmark
Fax: (+45) 86 13 17 69
e-mail: ajs@imf.au.dk and kima@imf.au.dk

August 16, 2001

Abstract

This article contributes with a very fast algorithm for solving the bicriterion shortest path problem. By imposing some simple domination conditions, we reduce the number of iterations needed to find all the efficient (Pareto optimal) paths in the network. We have implemented the algorithm and tested it with the Label Correcting algorithm. We have also made a theoretical argument of the performance of all the existing algorithms, in order to rank them by performance.

Included is a discussion on the structure of random generated networks, generated with two different methods, and of the characteristics of these networks.

Keywords: MCDM, MCIP, Bicriterion, Shortest Path, Random networks.

1 Introduction

The bicriterion shortest path problem (BSP) is one of the simplest problems in multicriterion integer analysis, but nevertheless also one of great importance in many applications. For example it is of interest to model transportation problems with more than one objective. Also, the BSP problem often occurs as a subproblem in other problems, for example in scheduling problems. It also occurs as a subproblem in models for transportation of hazardous materials, see Erkut *et al.* [4].

*Corresponding author.

Let us describe the problem. We have a *directed network* or a *digraph* $G = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{1, \ldots, n\}$ is the set of nodes and $\mathcal{A} = \{(i,j), (k,l), \ldots, (p,q)\}$ is a finite set of directed edges joining nodes in $\mathcal{N}$. Parallel edges are allowed. Each edge $(i,j) \in \mathcal{A}$ carries two attributes denoted by (c_{ij}, t_{ij}). For simplicity assume that c_{ij} is the cost using edge (i,j) and t_{ij} is the travel time from node i to node j (using the edge (i,j)). The objective is to find a "shortest" path from a particular node, the source node $s \in \mathcal{N}$, to another particular node, the terminal node $t \in \mathcal{N}$. Traditionally, the BSP problem is formulated as follows:

$$
\begin{array}{ll}
\min & F^1(x) = \sum_{(i,j)\in\mathcal{A}} c_{ij} \cdot x_{ij} \\
\min & F^2(x) = \sum_{(i,j)\in\mathcal{A}} t_{ij} \cdot x_{ij} \\
\text{s.t.} & \\
& \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} \quad - \quad \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ 0 & \text{if } i \neq s,t \\ -1 & \text{if } i = t \end{cases} \\
& x_{ij} \in \{0,1\}, \ \forall \ (i,j) \in \mathcal{A}
\end{array}
\tag{1}
$$

The constraints in (1) yield a directed path from source node s to terminal node t if one exists and the two objectives are to find the minimum cost $s-t$ path and the minimum travel time $s-t$ path, respectively. As it is highly unlikely to find a directed path from node s to node t which achieves both the minimum total cost and the minimum total travel time, we have to settle with something less, namely finding the set of efficient paths (see Definition 1) from node s to node t. Several approaches for doing this has been presented in the past, but we have only been able to find one attempt to compare the existing algorithms, see Huarng *et al.* [6], and they find the Label Correcting algorithm [2] to be the fastest.

The outline of the paper is as follows. In section 2 we describe the theory of the problem in question and give a theoretically based argumentation for the ranking of the existing algorithms. In section 3 we present the basic Label Correcting algorithm found in Brumbaugh-Smith *et al.* [2] along with our modified versions. In section 4 we discuss the structure of randomly generated digraphs for the BSP problem, because it turns out to have influence on the computational results. In section 5 we present our test results together with a comparison of the most promising methods.

2 The theory of bicriterion shortest path problems

Solving the BSP problem means finding the set of efficient paths from source node s to terminal node t. For basics in multiple criteria analysis see Steuer [8].

In order to make sure, that solutions do exist, we assume that the network is strongly connected. The definition of efficiency is as follows.

Definition 1 *A feasible solution x to (1) is* ***efficient*** *iff there does not exist another feasible solution $\bar{x}$ to (1) such that $(F^1(\bar{x}), F^2(\bar{x})) \leq (F^1(x), F^2(x))$ and $(F^1(\bar{x}), F^2(\bar{x})) \neq (F^1(x), F^2(x))$. Otherwise x is* ***inefficient****.*

Efficiency is defined in the decision space. There is a natural counterpart in the criterion space. The criterion space is denoted by Z and is given by $Z = \{z(x) \in R^2 | z(x) = (F^1(x), F^2(x)),\ x \text{ is feasible in } (1)\}$.

Definition 2 *$z(x) \in Z$ is a* ***non-dominated*** *criterion vector iff x is an efficient solution to (1). Otherwise $z(x)$ is a* ***dominated*** *criterion vector.*

It is well-known that the constraint set in (1) defines an integral polytope (the constraint-matrix is unimodular). Therefore, if the linear relaxation of (1) is solved, the set of extreme (supported) efficient paths is found. Unfortunately, there might be (and probably are) unsupported efficient paths as indicated in Figure 1.

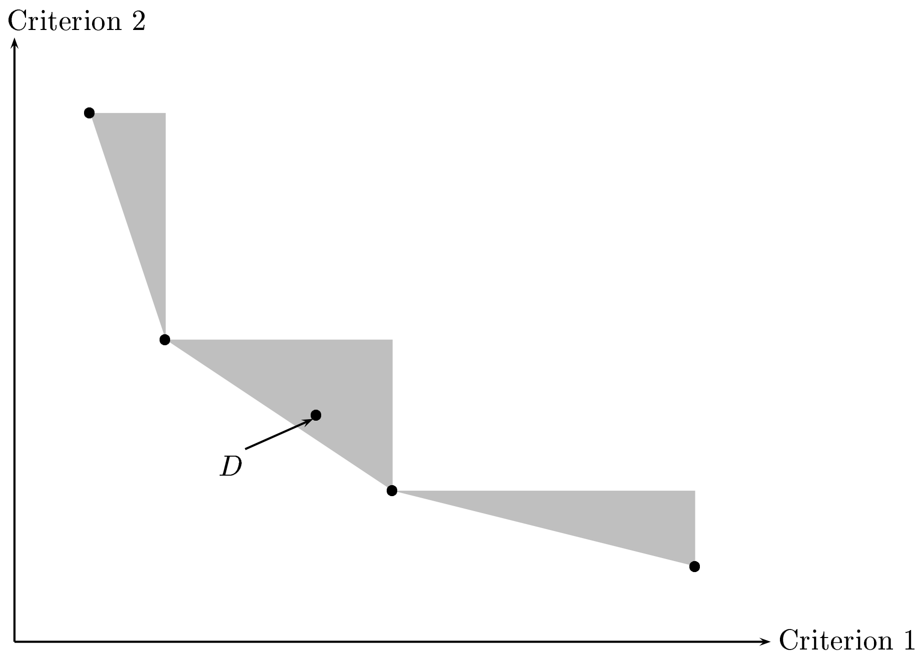

Figure 1: D is a non-extreme, non-dominated criterion vector.

Being interested in the set of efficient paths, it is not a satisfactory compromise just finding the set of supported efficient paths. It should, however, be noted that in practice the decision maker (DM) might be satisfied with only the set of extreme efficient paths. In fact, the DM will probably prefer to see the set of non-dominated values (criterion vectors).

Basicly, there are two approaches to the problem, namely some sort of path/tree handling procedure or some sort of node-labeling procedure. Climaco and Martin [3] and Mote *et al.* [7] fall in the path/tree handling category. Below we argue that this approach has disadvantages for the BSP problem. Hansen [5], Brumbaugh-Smith *et al.* [2] and the approach in this paper fall in the labeling category. Our ranking is found in Table 1.

Now we will use a small example to illustrate the complexity of the BSP problem. By evaluating two different sets of coefficients, we explain why the node-labeling approach is better than the path/tree handling procedure. For clarity remember that efficient paths are in the (high dimensional) decision space, and the non-dominated values are in the (two dimensional) criterion space. The network in Figure 2 has parallel edges. If we split the lower edges into two, where the edge-coefficients sum to $(2,1)$, the example is similar to one found in Hansen [5].

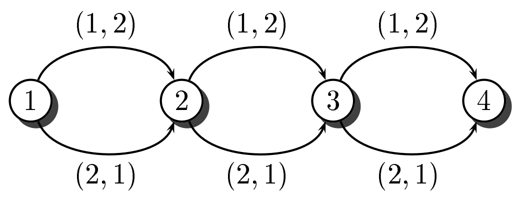

Figure 2: Illustration of complexity in efficient paths.

In Figure 2 there is a total of 8 paths. All paths are efficient, having the 4 non-dominated values $(3,6)$, $(4,5)$, $(5,4)$ and $(6,3)$. The upper path has the value $(3,6)$, then there are three paths having the value $(4,5)$, three paths having the value $(5,4)$ and the lower path has value $(6,3)$. We see that the efficient paths distribute among the non-dominated values, as in level 4 in Pascal's triangle, see Figure 3. Note that the network in Figure 2 has 4 nodes.

From this special case of the BSP problem we make two observations. The number of efficient paths grow exponentially in the number of nodes, namely $2^{|\mathcal{N}-1|}$, and the number of efficient paths are always greater than or equal to the number of non-dominated values, which is $|\mathcal{N}|$. In the above there are 8 efficient paths, and 4 non-dominated values.

Next we show an example, where the number of non-dominated values grow exponentially

Level 1				1			
Level 2			1		1		
Level 3		1		2		1	
Level 4	1		3		3		1

Figure 3: Pascal's triangle.

in the number of nodes. That is, all the efficient paths have distinct non-dominated values. We use this to conclude that the node-labeling algorithms have exponential complexity. Here we choose the edge coefficients, so that the sum of the smaller coefficients is less than the next. This is achieved by the following numbers, 2^i, $i = 0, 1, 2, \ldots, |\mathcal{A}| - 1$. For the example that is $1, 2, 4, 8, 16, 32$, and then we pair them from each end of the list as shown in Figure 4.

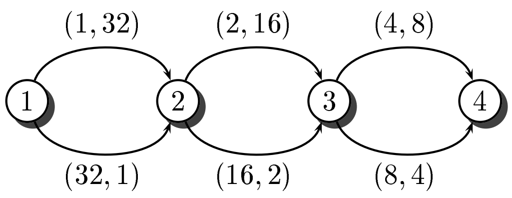

Figure 4: Illustration of complexity in Efficient values.

The 8 paths in the network in Figure 4 are all efficient having the following 8 non-dominated values: $(7, 56)$, $(11, 52)$, $(21, 42)$, $(25, 38)$, $(38, 25)$, $(42, 21)$, $(52, 11)$ and $(56, 7)$. By choosing the edge coefficients this way we get $2^{|\mathcal{N}-1|}$ non-dominated values.

We argue that the node-labeling algorithms will outperform the path/tree algorithms because the number of non-dominated values is always smaller than (or equal to) the number of efficient paths. A stronger argument is that the node-labeling algorithm only finds the list of non-dominated values at the terminal node, and not the actual efficient paths. After the Decision Maker chooses some favourite non-dominated value(s), we only have to backtrack for these particular efficient paths. As a final note we see that the amount of memory needed to store the labelsets, is much smaller than the memory needed to store all the efficient paths.

In the existing literature all algorithms, except perhaps the Parametric Approach by Mote *et al.* [7], have been proven slower than the Label Correcting approach. Comparisons are found in [2] and [6]. These algorithms are the Label Setting approach by Hansen [5], and the K'th shortest path approach by Climaco and Martin [3].

1	Skriver & Andersen	node-labeling
2	Brumbaugh-Smith & Shier [2]	node-labeling
3-4	Hansen [5]	node-labeling
3-4	Mote, Murphy & Olson [7]	path/tree handling procedure
5	Climaco & Martin [3]	path/tree handling procedure

Table 1: Existing BSP algorithms ranked by computational performance.

We will argue that the Parametric Approach will also be slower, due to the structure of the algorithm. The approach is to use the weighting method (see Steuer [8]) to find the efficient **extreme** paths, and then use backtracking of spanning trees to search for non-extreme efficient paths. The weighting method means solving LP problems, but for the shortest path problem that is done by Dijkstra's shortest path algorithm (or a similar algorithm). It turns out that Dijkstra's algorithm is actually a slower approach in practice than the Label Correcting routine, see section 5. On top of this comes the fact, that the weighting method of the Parametric Approach by far is faster than the backtracking part [7]. When we are backtracking, we might have to evaluate all the edges in all the spanning trees in the worst case, resulting in an exponentially growing number of comparisons. This structural disadvantage is also the case for Climaco and Martins [3] algorithm.

The conclusion is that the Parametric Approach and the K'th shortest path algorithm are slower than the Label Correcting approach, and this was also found by Huarng *et al.* [6]. Due to the structure of the backtracking part of the Parametric Approach, we also believe that it is slower than Hansen's Label Setting algorithm, especially for networks with negatively correlated objectives, but this has not been tested.

Hansen's algorithm is a label setting scheme with an exponentially worst case behaviour. It uses four sets of labels instead of only two, as in the Label Correcting approach, and it makes more set comparisons. We therefore rank it below the Label Correcting approach. The Label Correcting approach uses the well-known fact that all efficient paths pass through efficient subpaths.

3 Algorithm with preprocessing routine

In this section we describe the label-correcting algorithm proposed by Brumbaugh-Smith *et al.* [2]. The description is followed by a couple of suggestions for improvements, which give rise to a refinement of the Brumbaugh-Smith algorithm.

The theory and idea behind this is explained in the following sections, but for simplicity we start with some notation.

3.1 Notation

$D(i)$: Set of labels at node i. Each label is a 2-tuple containing cost and time

s : source node

t : terminal node

Labeled : Set of nodes to be checked

$Len(i,j)$: The edge-length from node i to node j (two attributes cost and time)

$out(i)$: Set of edges having their tail in node i

$Merge(A,B)$: $A \cup B \setminus$ (dominated elements in $A \cup B$)

$c^*(i)$: cheapest path from node i to node t

$t^*(i)$: fastest path from node i to node t

$c^{min}(i)$: cheapest path from node s to node i

$t^{min}(i)$: fastest path from node s to node i

$\hat{c}$: upper bound on cost, corresponding to $t^{min}(t)$

$\hat{t}$: upper bound on time, corresponding to $c^{min}(t)$

3.2 The Brumbaugh-Smith Algorithm

The algorithm below is taken directly from Brumbaugh-Smith *et al.* [2]. The boxed part of the algorithm is the time consuming part we try to avoid when it is not needed. The FIFO principle is used to select nodes from the set Labeled as recommended in Brumbaugh-Smith *et al.* [2].

Initialize:

$D(s) = \{(0,0)\}$;

`Labeled=`$\{\ s\ \}$ `;`

Routine:

`while Labeled` $\neq \emptyset$

`choose` i `from Labeled (FIFO principle);`

`Labeled=Labeled-`$\{\ i\ \}$ `;`

`for` $j \in out(i)$

```
      D_M(j) = Merge(D(j), D(i) + len(i,j));
      If D_M(j) ≠ D(j) then
        D(j) = D_M(j);
        If j not in Labeled then (avoids double labelling)
          Labeled = Labeled + { j } ;
        end If;
      end If;
  end for;
end while;
```

In this algorithm the Merge operation in the box uses the main part of the computational effort. Our intention is to discard "expensive" edges before these operations are carried out in order to reduce computation time. The means being inducing some simple domination conditions on the edge-candidates in order to discard "expensive" edges as soon as possible. The Merge operation returns the labels in an ordered set as described in section 3.3.1. The Merge operation implemented in this paper is the "modified Merge" operation found in Brumbaugh-Smith *et al.* [2]. This operation is in linear time as a function of the sizes of the two sets to be merged.

It should be noted that the algorithmic structure of the Brumbaugh-Smith algorithm is somewhat similar to Dijkstra's shortest path method, except the nodes can reenter in the set Labeled. This suggests that the performance is similar. In the remainder of this paper we refer to the Brumbaugh-Smith algorithm as **brum**.

3.3 The improvements

We have two suggestions for improvements that are both based on the idea of omitting "expensive" edges before the **box** in the algorithm. At each iteration in the routine, we are looking at an edge (i, j) from some node i to another node j, see Figure 5.

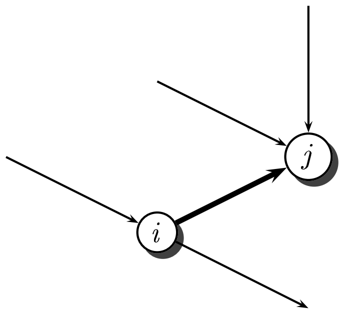

Figure 5: Evaluating the (i, j)-edge.

The first improvement is a fast predomination check, which rules out "expensive" edges by considering the present set of labels. This condition is implemented in two distinct ways as -described in sections 3.4 and 3.5. The first implementation uses initialization with Dijkstra's shortest path method to set bounds on all labels, and the second procedure sets the bounds during the routine.
The second improvement is inspired by an article by Tung and Chew [9]. The idea is to initialize node information from the terminal node in order to find the cheapest and fastest paths from an intermediate node to the terminal node. This initialization also finds some upper bounds on the two objectives. If the present best label at node i, plus the cost/time of the (i,j)-edge, plus the least cost/time for the remainder of the (j,t)-path exceeds the upper bounds, the edge (i,j) can be left out of further consideration.

We will argue that initializing using Dijkstra's shortest path method is too slow. This is supported by the computational results described in section 5. The problem with the initialization is that the bounds set on the labelsets are too loose. The bounds set during the routine is better, and can therefore discard more "expensive" edges. The computational results shown in section 5 support this.

3.3.1 Condition I

Consider again the two particular nodes, i and j, and the set of labels $D(i)$ and $D(j)$ at these two nodes. Assume that the two labelsets are non-empty, and that
$D(i) = \{(c_1(i), t_1(i)), \cdots, (c_k(i), t_k(i))\}$ and $D(j) = \{(c_1(j), t_1(j)), \cdots, (c_q(j), t_q(j))\}$ with
$c_1(i) < c_2(i) < \cdots < c_k(i)$ and $t_1(i) > t_2(i) > \cdots > t_k(i)$

$c_1(j) < c_2(j) < \cdots < c_q(j)$ and $t_1(j) > t_2(j) > \cdots > t_q(j)$

We are now looking at the edge from node i to node j. Consider the two distinct but similar situations:

- Assume that $c_1(i) + len(i,j)_c \geq c_q(j)$. In this case we have:

  $c_1(j) < c_2(j) < \cdots < c_q(j) \leq c_1(i) + len(i,j)_c < \cdots < c_k(i) + len(i,j)_c$

  $t_1(j) > t_2(j) > \cdots > t_q(j) \ ? \ t_1(i) + len(i,j)_t > \cdots > t_k(i) + len(i,j)_t$

  So if $t_k(i) + len(i,j)_t \geq t_q(j)$, then the set $D(i) + len(i,j)$ is dominated by the set $D(j)$. In fact the set $D(i) + len(i,j)$ is dominated by the last label q of $D(j)$. Therefore we can discard the edge between i and j, and proceed to the next edge, because a merge of the two sets will return the set $D(j)$ unchanged.

- Assume that $t_k(i) + len(i,j)_t \geq t_1(j)$. In this case we have:

  $c_1(i) + len(i,j)_c < \cdots < c_k(i) + len(i,j)_c \ ?\ c_1(j) < c_2(j) < \cdots < c_q(j)$

  $t_1(i) + len(i,j)_t > \cdots > t_k(i) + len(i,j)_t \geq t_1(j) > t_2(j) > \cdots > t_q(j)$

  So if $c_1(i) + len(i,j)_c \geq c_1(j)$, then the set $D(i) + len(i,j)$ is dominated by the set $D(j)$, because it is dominated by the first label of $D(j)$.

The above observations give rise to the following pseudo-code:

If $c_1(i) + len(i,j)_c < c_q(j)$ `then end if; (because` (i,j) `is promising)`
`else (That means` $\geq$`)`
  `If` $t_k(i) + len(i,j)_t < t_q(j)$ `then end if; (because` (i,j) `is promising)`
  `else remove` j `from` $out(i)$`; (because` (i,j) `is dominated)`
`If` $t_k(i) + len(i,j)_t < t_1(j)$ `then end if; (because` (i,j) `is promising)`
`else (That means` $\geq$`)`
  `If` $c_1(i) + len(i,j)_c < c_1(j)$ `then end if; (because` (i,j) `is promising)`
  `else remove` j `from` $out(i)$`; (because` (i,j) `is dominated)`

The case of alternative solution possibilities is discarded in the strict inequalities. The actual paths, alternative solutions or not, can be found by a simple backtracking algorithm. This way we only find the path(s) that have the "best" cost/time labels (viewed from the terminal node t by the DM).

Another interesting case is when we look at the opposite conditions of the above. This implies that all labels at node j is dominated by the paths via node i, and therefore can be replaced by a new set of labels with the simple calculation $D(j) = D(i) + len(i,j)$. This operation we will call *overtaking*. However, not surprisingly the number of times we can "overtake" is small, because relatively good bounds are set as we proceed through the network. Therefore "overtaking" is used only when node j has not yet been labeled.

Notice that the sets of labels are expected to be small in the beginning (1-3 labels), but larger as we approach node t.

3.3.2 Condition II

To use the second improvement it is necessary to use Dijkstra's shortest path method starting at node t, and with all edges reversed, in the initialization. When used with both cost and time, we find the cheapest and fastest path from any node i to the terminal node t. These values are denoted $c^*(i)$ and $t^*(i)$. The upper bounds on cost and time are denoted $\hat{c}$ and $\hat{t}$, and are found when initializing with Dijkstra's shortest path method is used to find the cheapest and fastest paths.

This gives the following pseudo-code:
`If` $c_1(i) + len(i,j)_c + c^*(j) \geq \hat{c}$ `then remove` j `from` $out(i)$
`If` $t_k(i) + len(i,j)_t + t^*(j) \geq \hat{t}$ `then remove` j `from` $out(i)$

In the next sections we discuss how to implement Condition I. In section 5 we argue that Condition II will be too slow due to the initialization with Dijkstra's shortest path method as mentioned in the beginning of this section.

3.4 Algorithm 1 - Initializing with Dijkstra's shortest path procedure

We implement the Brumbaugh-Smith algorithm together with condition I. Condition I described above is first implemented using Dijkstra's shortest path method to initialize the algorithm, and to set bounds on the labelsets. We refer to this algorithm as **alg1**. The pseudo-code is as follows:

Initialize:
$D(s) = \{(0,0)\}$;
`Use Dijkstra's algorithm to minimize cost`
`Use Dijkstra's algorithm to minimize time`
`Labeled=`$\{$`1, 2, ... , n` $\}$ `; (all nodes need to be examined during the routine)`
`SLIM(`$D(i), \forall i \in$ `Labeled);`

After having used Dijkstra's algorithm two times, all nodes have two labels, and these labels are also the bounds corresponding to the two elements $(\hat{c}(i), t^{min}(i))$ and $(c^{min}(i), \hat{t}(i))$. There might be some duplicate labels, because the labels set by Dijkstra's shortest path method, minimizing cost and time, might be the same, especially in the beginning of the network. If the two labels are equal, one of them is deleted. We refer to this procedure as SLIM in pseudo-code.

Routine:
`while Labeled` $\neq \emptyset$
  `choose` i `from Labeled (FIFO principle);`
  `Labeled=Labeled-`$\{$ i $\}$ `;`
  `for` $j \in out(i)$
`Condition I`
    `If` $c_1(i) + len(i,j)_c < c_q(j)$ `then end if; (because` (i,j) `is promising)`

```
      else (That means ≥)
        If t_k(i) + len(i,j)_t < t_q(j) then end if; (because (i,j) is promising)
        else remove j from out(i); (because (i,j) is dominated)
      If t_k(i) + len(i,j)_t < t_1(j) then end if; (because (i,j) is promising)
      else (That means ≥)
        If c_1(i) + len(i,j)_c < c_1(j) then end if; (because (i,j) is promising)
        else remove j from out(i); (because (i,j) is dominated)
Condition I ended
The box is only carried out if the (i,j)-edge looks promising
+-------------------------------------------------------------+
|     D_M(j) = Merge(D(j), D(i) + len(i,j));                  |
|     If D_M(j) ≠ D(j) then                                   |
|       D(j) = D_M(j);                                        |
|       If j not in Labeled then (avoids double labelling)    |
|        Labeled = Labeled + { j } ;                          |
|       end If;                                               |
|     end If;                                                 |
+-------------------------------------------------------------+
   end for;
end while;
```

The computational performance of the algorithm can be seen in Table 4.

3.5 Algorithm 2 - A direct approach

This implementation of Condition I is without initialization, but with "overtaking" of nonlabeled nodes. The algorithm is referred to as **alg2**. The pseudo-code is as follows:

Initialize:

$D(s) = \{(0,0)\}$;

```
Labeled={ s } ;
```

Routine:

```
while Labeled ≠ ∅
   choose i from Labeled (FIFO principle);
   Labeled=Labeled-{ i } ;
   for j ∈ out(i)
     If j not in Labeled then D(j) = D(i) + len(i,j) (``overtaking'')
     else
Condition I
      If c_1(i) + len(i,j)_c < c_q(j) then end if; (because (i,j) is promising)
      else (That means ≥)
        If t_k(i) + len(i,j)_t < t_q(j) then end if; (because (i,j) is promising)
```

```
      else remove j from out(i); (because (i,j) is dominated)
     If t_k(i) + len(i,j)_t < t_1(j) then end if; (because (i,j) is promising)
     else (That means ≥)
      If c_1(i) + len(i,j)_c < c_1(j) then end if; (because (i,j) is promising)
      else remove j from out(i); (because (i,j) is dominated)
Condition I ended
The box is only carried out if there is no ``overtaking'' or if the (i,j)-edge
looks promising.
```

```
     D_M(j) = Merge(D(j), D(i) + len(i,j));
     If D_M(j) ≠ D(j) then
       D(j) = D_M(j);
       If j not in Labeled then (avoids double labelling)
         Labeled = Labeled + { j } ;
       end If;
     end If;
```

```
  end for;
end while;
```

4 Random networks

In this section we compare two different approaches for generating random networks. The reason being that the structure of the random networks has a large effect on the computational results.
First we discuss how NETGEN[1] works and what problems this gives. Then we introduce our own random network generator NETMAKER[2] and explain in detail how it works. Finally, we compare the two approaches computationally to illustrate the differences.

Our first approach was to use NETGEN to generate random networks (without the cost/time coefficients). On these networks we ran a small program to generate the cost and time coefficients. These coefficients are generated in the same way as in NETMAKER described below.

We generated ten random networks with NETGEN, each having 100 nodes and 900 edges. In average, there were only 7.5 non-dominated values (see Table 2), and four of the networks had only 2 or 3 non-dominated values.
At first it seemed a little strange that the average number of non-dominated values generated with NETGEN was so low. In an effort to explain this, we drew all the efficient

[1] shareware software found on the Internet and used in Huarng *et al.* [6]
[2] available in C++ on the webpage http://www.imf.au.dk/~ajs/

paths in a NETGEN generated network. The structure shown in Figure 6 was found.

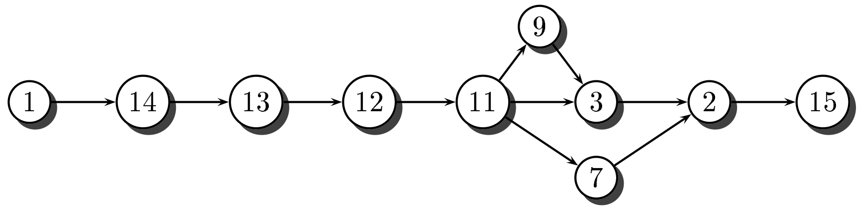

Figure 6: Structure of efficient paths in NETGEN network with 15 nodes.

The network is being run through on a single efficient subpath, and then somewhere there is a few alternative subpaths between a few nodes, before the paths again use the same subpath to the terminal node. The efficient paths are not spread out through the network. This network structure is due to the generation of a Hamiltonian cycle in NETGEN that is deterministic, and because the generation of random edges are uncontrolled. Before any random edges are generated, a Hamiltonian cycle is made as shown in Figure 7, to make sure that the network is strongly connected.

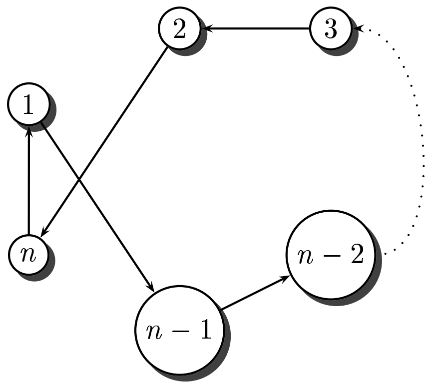

Figure 7: Hamiltonian cycle in NETGEN.

Because of this deterministic structure of the Hamiltonian cycle, a single efficient subpath is used, in the beginning as well as in the end of the network.

We find this an unrealistic structure for real life problems. For this reason we developed the NETMAKER program in order to generate alternative efficient paths that run through the whole network having a structure similar to that shown in Figure 8. This structure is actually found in the NETMAKER networks.

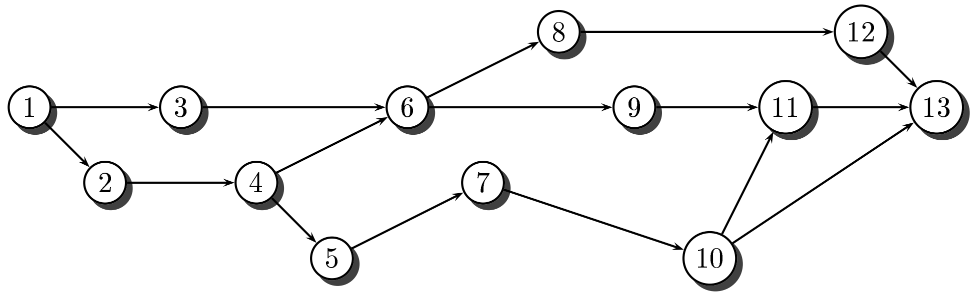

Figure 8: Structure of efficient paths in NETMAKER networks.

Next we describe how NETMAKER works. First a **random** Hamiltonian cycle starting at the source node s is generated in order to secure that the network is strongly connected. Then we uniformly generate a random number of edges out of each node. This random number of edges belongs to a certain interval, say 1 to 3 edges, to control the total number of edges. These edges are only allowed to reach a certain number of nodes forward and backwards. This omits paths with very few edges, unless they are generated in the random Hamiltonian cycle. This edge interval is essential to get the structure of Figure 8, with efficient paths spread out through the network.

To illustrate this, we assume that we are allowed to generate between 1 and 3 edges out of node 5, within a node-interval of 6 nodes. We want node 5 in the middle of the interval, so the edges may reach 3 nodes in each direction. From node 5 these edges are allowed to go into nodes 2, 3, 4, 6, 7 and 8. The 6 possible edges are shown in Figure 9.

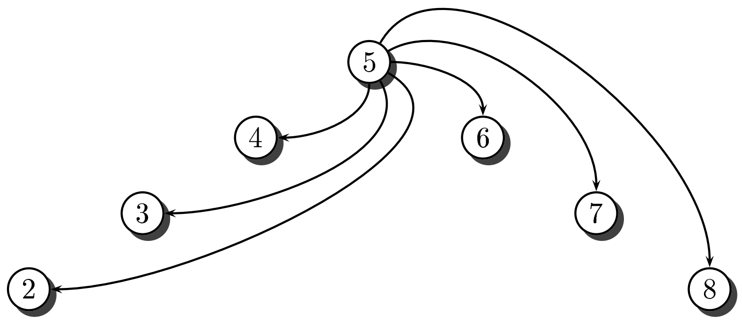

Figure 9: The 6 possible random edges in which 1, 2 or 3 must be picked.

The cost/time coefficients are generated negatively correlated so that one coefficient is an integer between 1 and 33, and the other is between 67 and 100. This coefficient generating approach was found in Andersen, Jörnsten and Lind [1]. We use negatively correlated objectives because it is accepted to be most realistic and interesting, and because it implies more efficient paths [2].

In Table 2 it is seen that the number of non-dominated values is much higher in the networks generated by NETMAKER. One could think that this is only because of the random Hamiltonian cycle in NETMAKER, but as illustrated in Table 3 the edge intervals are important.

generator	# nodes	# edges	non-dominated values
NETGEN	100	900	7.5
NETMAKER	100	$\simeq$ 900	13
NETGEN	50	250	4.7
NETMAKER	50	$\simeq$ 250	8.9

Table 2: Average number of non-dominated values in NETGEN and NETMAKER

To get an idea of the number of non-dominated values in networks generated with NETMAKER compared with the number of non-dominated values in networks generated with NETGEN we generated networks with 200 nodes. The results are shown in Table 3. Each column in table 3 presents the number of outgoing edges for each node in the networks generated. The 3-6 column, for example, indicates that from each node between 3 and 6, outgoing edges are generated, 6 being the max. The rows in the table give the interval between which these edges are allowed to go. The first row ($2 \cdot \max$) shows that the edges generated are distributed uniformly in an interval of "$2 \cdot \max$" length around each node where "max" is the maximum number of edges generated. Consider the 3-6 column and the ($2 \cdot \max$) row. From a particular node, say 10, between 3 and 6 outgoing edges are generated. These edges can go into nodes with numbers between 4 and 16 (the interval length is 2 times the maximum number of edges generated, $2 \cdot 6 = 12$). As another example consider the 7-15 column and the ($3 \cdot \max$) row. From a particular node, say 25, between 7 and 15 outgoing edges are generated. These edges can go into nodes with numbers between 3 and 47 (the interval length is approximately 3 times the maximum number of edges generated).

In Table 3 we have found that the average number of non-dominated values depend on the degree of each node and on the edge interval size. All cells in the tables are with a sample size of 10 networks.

200 nodes	1-3	2-4	3-6	5-10	7-15	10-20
$2 \cdot$ max	3.5	5.5	10.2	12.4	13.1	14.3
$3 \cdot$ max	3.9	4.8	9.6	12.7	13.8	13.9
$4 \cdot$ max	3.1	4.9	7.9	12.9	13.3	14.7
$8 \cdot$ max	3.5	5.5	8	10.8	12.2	15
no restriction	2.5	4.3	6.4	8	9.8	10
200 interval	3.4	5.1	7.4	9.9	9.2	13.1
NETGEN	2.7	4.8	6.7	7.9	8.2	10.3

Table 3: Average number of non-dominated values generated by NETMAKER

It is of no surprise that the average number of non-dominated values increases with the total number of edges. The total number of edges varies around its mean being the average number of edges created at each node times the number of nodes. For example, 2-4 edges implies an average of 3 edges per node, and with 200 nodes this results in approximately 600 edges in each network of this kind. This is the fixed number of edges generated for the NETGEN networks in the last row of Table 3 for the 2-4 edges case. We see that NETGEN generates the same number of non-dominated values that NETMAKER does without interval restrictions. This means that when you run NETMAKER with no interval restriction, the probability of a short path is high, as explained with NETGEN.

The next observation is on the interval size. We claimed earlier that the number of non-dominated values will be larger if paths using very few edges are omitted. This claim holds, as the number of non-dominated values is larger when there is a restriction on the interval size. Since the interval length itself is not so important, we have used the $8 \cdot$ max interval length for all the networks in section 5.

As mentioned earlier, the structure of the random networks shown in Figure 6 is an unrealistic structure for testing shortest path algorithms, and it also favours one type of algorithm, namely the Parametric Approach [7]. The reason for this is that the Label Correcting algorithm (brum) has to evaluate all nodes at least once, and thereby evaluates all edges. Therefore the computational effort is very dependent on the size of the network, even in networks with only one efficient path. The Parametric Approach uses simple objective weighting in the first phase, and then backtracking of spanning trees as explained in section 2. But in case of a large network with only one or two efficient paths there is little backtracking, and the algorithm will perform well. But if the network has many efficient paths and perhaps negative correlation between the objectives, there is much more backtracking to be done, and this is what takes time in the Parametric algorithm [7].

Mote *et al.* [7] have computational results in their article showing that with negatively correlated objectives, their algorithm has cpu-times similar to those of Hansen's label setting approach [5]. This illustrates how much cpu-time is consumed by the backtracking part, if there are many efficient paths.

5 Computational results

We have tested the two algorithms described in sections 3.4 (alg1) and 3.5 (alg2) together with the Brumbaugh-Smith algorithm (brum) of section 3.2. All algorithms are implemented in C++, and can be found on the homepage http://www.imf.au.dk/~ajs. We have used an HP 9000 series computer with a single processor. For each size of network we have used 10 random networks, so in Table 4 we have used a total of 50 networks.

In the previous section, we argued that NETMAKER generates reasonably random networks for testing bicriterion shortest path algorithms. In this section we compare the Brumbaugh-Smith approach and the modified versions of section 3.4 (alg1) and section 3.5 (alg2).

# nodes	brum	alg1 init	alg1 routine	alg1 total	alg2
100	2.52	5.22	2.11	7.33	1.84
200	18.27	41.32	16.54	57.86	13.59
300	44.37	139.76	40.81	180.57	35.11
400	76.26	327.98	70.52	398.50	58.29
500	133.22	640.29	123.96	764.25	108.33

Table 4: Cpu-times for brum, alg1 and alg2 when the number of edges are between 2 and 4 at each node.

In Table 4 we see that the initialization phase in alg1, where Dijkstra's shortest path algorithm is run through twice, takes more than double the amount of cpu-time used by brum. Our implementation of Dijkstra's shortest path algorithm is seen to be a little slower than the Label Correcting routine. Remember that they are expected to be fairly similar.

The difference in cpu-time between the alg1 routine part and alg2 occurs because all nodes in the alg1 routine part has a label from initialization so there is no "overtaking".

The slow initialization times are the reason why Condition II in section 3.3.2, suggested by Tung *et al.* [9] is not being implemented. It requires that Dijkstra's shortest path method is run through twice. The brum algorithm is always outperformed by the alg2.

It can also be seen that running Dijkstra's shortest path method twice takes somewhat the double cpu-time as running brum. This supports our argument from section 3.3, that the Label Correcting algorithm has the same computational performance as Dijkstra's.

The rest of the comparisons are done with only the brum and alg2 algorithms. The objective is to evaluate the effectiveness of Condition I on networks with different characteristics.

# nodes	brum	Merges	alg2	Condition I's	% Merges in alg2	% cpu-time
200	9.01	761.30	4.12	208.40	46.49	45.76
400	40.38	1615.20	20.96	407.20	50.09	51.91
600	92.96	2502.00	51.40	578.15	52.95	55.29
800	187.05	3385.00	111.82	757.65	54.01	59.78
1000	280.61	4668.20	162.52	970.90	57.80	57.92

Table 5: Cpu-times, number of Merges and number of Condition I's for brum and alg2 when the number of edges are between 1 and 3 at each node.

The first comparison is made on a thin network, where the average number of edges is only two times the amount of nodes. The results are shown in Table 5, and the overall conclusion is that alg2 is considerably faster than brum. On these thin networks, Condition I is active in about 25 % of the set comparisons. On top of this the "overtake" procedure labels ($\#\text{nodes} - 1$) times, this being the number of times we look at a node with an empty labelset. If we add the number of nodes to the number of Condition I's, we get half the number of merges carried out in the brum algorithm. This explains why the cpu-time is half, and supports that the boxed part of the algorithms is the computationally heavy part.

It can also be seen, that as the number of nodes increases, the fraction of Condition I's decreases. This is due to the fact that the probability of the Condition I being fulfilled decreases as the labelsets increase. The labelsets increase in size as we move towards the terminal node, and in the larger networks the average number of non-dominated values is a little higher and therefore the labelsets are bigger. The Condition I is more often fulfilled in the first half of the merges, while the labelsets are fairly small. As expected alg2 performs very well on thin networks, because of the small size of the labelsets.

Next we look at less thin networks with an average number of 3 edges per node. The results are shown in Table 6, and as expected the fraction of Condition I's has dropped. Because the cpu-time saved is fairly proportional to the number of Condition I's and "overtakes",

# nodes	brum	Merges	alg2	Condition I's	% Merges in alg2	% cpu-time
200	18.34	1399.65	12.16	303.25	64.12	66.29
300	45.37	2221.4	31.76	428.45	67.25	70.00
400	80.43	3080.7	58.14	487.35	71.23	72.28
500	129.77	4006.65	96.91	652.10	71.27	74.68
800	336.65	6801.80	245.77	933.95	74.52	73.00

Table 6: Cpu-times, number of Merges and number of Condition I's for brum and alg2 when the number of edges are between 2 and 4 at each node.

alg2 only performs about 25-35 % better than the brum algorithm for networks with this density (and this size).

# nodes	brum	Merges	alg2	Condition I's	% Merges in alg2	% cpu-time
100	12.59	2796.1	11.05	251	87.48	87.76
200	79.55	6055.40	73.50	284.4	92.02	92.40
300	195.48	9680.60	183.55	346.45	93.33	93.90
400	349.04	13733.30	329.83	430.25	93.96	94.50
500	589.84	17943.05	558.87	463.40	94.64	94.75

Table 7: Cpu-times, number of Merges and number of Condition I's for brum and alg2 when the number of edges are between 7 and 15 at each node.

For the thick networks of Table 7 with an average of 11 edges per node, we see that the fraction of Condition I's and "overtakes" is down to 5-12 %. The cpu-times are again proportionately faster as well. This Table illustrates that even in thick networks there are still cpu-time saved by imposing the condition. We therefore conclude that the cost in cpu-time of checking the condition is close to zero.

A little investigation revealed that for random networks, the number of merges is (almost) a linear function of the number of edges. For small networks the number of merges is double the amount of edges, and for larger networks the number of merges was found to be three times the number of edges.

6 Concluding remarks

We have investigated both the structure of random networks for the BSP problem, and the performance of the existing algorithms. Only the most promising algorithm so far, namely the Label Correcting algorithm has been implemented here. However, with reference to other articles, we argue that this approach is indeed the fastest. We have also imposed a condition to be checked during the routine, that saves up to 50 % cpu-time. Thus we conclude that even large BSP problems can be solved to optimality in reasonable time. Well-known methods for choosing among the non-dominated solutions can then be applied.

References

[1] K.A. Andersen, K. Jörnsten, and M. Lind. On bicriterion spanning trees: An approximation. *Computers and Operations Research*, 23:1171–1182, 1996.

[2] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43:216–224, 1989.

[3] J.C.N. Climaco and E.Q.V. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11:399–404, 1982.

[4] E. Erkut and V. Verter. Modeling of transport risk for hazardous materials. *Operations Research*, 46:625–642, 1998.

[5] P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, editors, *Multiple criteria decision making: theory and applications*, Lecture Notes in Economics and Mathematical Systems 177, pages 109–127. Springer-Verlag, Heidelberg, 1980.

[6] F. Huarng, P.S. Pulat, and L. Shih. A computational comparison of some bicriterion shortest path algorithms. *J. Chinese Inst. Indust. Engrs.*, 13:121–125, 1996.

[7] J. Mote, I. Murthy, and D.L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53:81–92, 1991.

[8] R.E. Steuer. *Multiple criteria optimization: Theory, Computation, and Application.* Wiley, New York, 1986.

[9] C.T. Tung and K.L. Chew. A bicriterion pareto-optimal path algorithm. *Asia-Pacific Journal of Operations Research*, 5:166–172, 1988.

The Bicriterion Semi-obnoxious Location (BSL) Problem Solved by an ϵ-Approximation

ANDERS J.V. SKRIVER AND KIM ALLAN ANDERSEN*
Department of Operations Research
University of Aarhus, building 530
Ny Munkegade
DK - 8000 Århus C
Denmark

August 16, 2001

Abstract

Locating an obnoxious (undesirable) facility is often modeled by the maximin or maxisum problem. But the obnoxious facility is often placed unrealistically far away from the demand points (nodes), resulting in prohibitively high transportation cost/time. One solution is to model the problem as a semi-obnoxious location problem.

Here we model the problem as a bicriterion problem, not in advance determining the importance of the obnoxious objective compared to the cost/time objective.

We consider this model for both the planar and the network case. The two problems are solved by an approximation algorithm, and the models are briefly compared by means of a real-life example.

Keywords: Multiple criteria analysis, Semi-obnoxious, Location, Planar, Networks.

1 Introduction

In the two traditional single facility location problems, a new facility is located (placed) so as to minimize transportation costs (minisum), or as to minimize the distance to the farthest customer (minimax). In the minisum problem we sum all the distances between the new facility and the customers, multiplied by a weight depending on the individual customer. In the minimax problem we minimize the largest weighted distance. The minisum model can be relevant when locating a warehouse and the minimax model can be used to locate a fire station. These models are presented in Love *et al.* [12] and Francis *et al.* [8], both including many references. The obnoxious location problem is a more recent class of problems, where the two most common are the maxisum and maximin models. When locating an obnoxious (undesirable) facility the goal is to place it as far from the

*Corresponding author. Email: kima@imf.au.dk

existing facilities (demand points, customers) as possible. See Erkut and Neuman [7] or Carrizosa and Plastria [5] for a review.

There is little literature combining the desirable and the obnoxious facility location models. In this paper we model the combined problem as a Bicriterion Semi-obnoxious Location (BSL) problem. One objective function is obnoxious and one is desirable. We also consider both the network case and the planar case of the problem. In biobjective optimization our goal is to find the set of efficient solutions. These solutions are such that there does not exist another solution that has a better value in one objective without having a worse value in the other objective. The concept of efficient solutions is the same as Pareto optimal solutions. In the network case, where the demand points are nodes in a network and we try to locate the new facility in a node or on an edge, we have found no references, but ongoing research is presented in Hamacher *et al.* [9]. In the planar case, where the feasible locations are in $\mathbb{R}^2$, we have found only three references, namely two papers by Brimberg and Juel, [1] and [2], and a paper by Carrizosa *et al.* [4].

In the bicriterion model, developed in the first paper by Brimberg and Juel [1], the first objective is the minisum objective and the second objective (the obnoxious criterion) is the minisum objective, where the Euclidean distance is raised to a negative power. It is proposed to solve the problem (finding the efficient solutions) in two steps. First a convex combination with parameter $\lambda \in [0, 1]$ of the two objectives (weighting method, Steuer [14]) is formed. The resulting objective is neither convex nor concave. By varying λ a trajectory of efficient solutions may be determined. In the paper an algorithm based on this is outlined. A numerical example is presented.

In the second paper by Brimberg and Juel [2] a different bicriterion model is considered. In this model the first objective is again the minisum objective, but the second objective (obnoxious) is now the maximin objective. They present two different solution methods for this model, but only one of them is guaranteed to find the complete set of efficient solutions.

In the bicriterion model developed in the third paper by Carrizosa *et al.* [4], the first objective (the obnoxious criterion) is modeled as the maxisum, and the second objective is modeled as the minisum problem. A solution procedure based on the BSSS (Big Square Small Square) approach is suggested. The procedure finds an approximation of the set of efficient solutions but no computational experience is reported. It should also be mentioned, that the approximation is in value space, and not in decision space.

The theory of the planar and network models is quite different, and the two models are not often compared, even though they try to describe the same real-life problem. We apply the two models on a real-life example in Section 4.

Next we present the basic model for the (BSL) problem. We assume that there are n existing facilities (demand points). In the planar case they are denoted $a_j = (a_{j1}, a_{j2})$, $j = 1, \dots, n$. In the network case they are denoted $v_1, v_2, \dots, v_n$. We want to place a new facility at location x in order to minimize both the (transportation) costs and the obnoxiousness. Let S denote the set of feasible solutions, $f(x)$ the obnoxious objective function and $g(x)$ the cost objective function. The general model looks as follows:

$$\begin{array}{ll} \min & f(x) \\ \min & g(x) \\ \text{s.t.} & \\ & x \in S \end{array} \tag{1}$$

We assume f depends negatively on the distance function and g depends positively on the distance function. This means, when we increase the distance between the new facility and an existing facility, this will have a decreasing effect on f and an increasing effect on g, e.g. less obnoxiousness but higher transportation costs.

Definition 1 *A feasible solution x to (1) is **efficient** iff there does not exist another feasible solution $\bar{x}$ to (1) such that $f(\bar{x}) \le f(x)$, $g(\bar{x}) \le g(x)$ and $(f(\bar{x}), g(\bar{x})) \ne (f(x), g(x))$. Otherwise x is **inefficient**.*

Efficiency is defined in the decision space. There is a natural counterpart in the criterion space. The feasible region in criterion space is denoted by $\mathcal{Z}$ and is given by $\mathcal{Z} = \{z(x) \in R^2 | z(x) = (f(x), g(x)),\ x \text{ is feasible in } (1)\}$.

Definition 2 *$z(x) \in \mathcal{Z}$ is a **nondominated** criterion vector iff x is an efficient solution to (1). Otherwise $z(x)$ is a **dominated** criterion vector.*

For a textbook introduction to multicriteria analysis see Steuer [14] or more recently Ehrgott [6]. We note that several efficient solutions may correspond to the same nondominated criterion vector.

As mentioned we consider two cases of the problem. The planar case, denoted the BSPL problem, where the feasible solutions form a region in the plane, and the network case, denoted the BSNL problem, where the set of demand points are vertices in a network.

The BSPL problem is solved using the BSSS method described by Hansen *et al.* [10], and we use the idea of this method to solve the BSNL problem as well. The method is described in Section 2.1 for the planar case and in Section 3.1 for the network case.

The remaining part of the paper is organized as follows. In Section 2 we describe the BSPL problem and the solution approximation algorithm, and in Section 3 the BSNL problem and its solution method is described. In Section 4 an application of the two models is presented. Section 5 contains the conclusions.

2 The planar case : The BSPL problem

The BSPL problem is formulated in the following way. There are n facilities (demand points) located at points $a_1, a_2, \dots a_n$, and the objective is to locate a semi-obnoxious facility at x so as to minimize a weighted sum of the distances raised to a negative power, and to minimize the weighted sum of the distances between the existing facilities and the new facility. The first criterion $f(x)$ may be thought of as a pollution effect and the second criterion $g(x)$ as transportation costs. This model was first introduced in Brimberg and Juel [1], where a discussion of the objective functions can also be found.

$$\begin{aligned} \min \quad & f(x) = \sum_{j=1}^{n} w_j^1 (\| x - a_j \|_{p_1})^{-b}, \quad b > 0 \\ \min \quad & g(x) = \sum_{j=1}^{n} w_j^2 \| x - a_j \|_{p_2} \\ \text{s.t.} \quad & \\ & x \in S \end{aligned} \tag{2}$$

where $\| x - a_j \|_p = (|x_1 - a_{j1}|^p + |x_2 - a_{j2}|^p)^{1/p}$ be the usual l^p norm, $p \geq 1$.

We prefer this obnoxious function, because it minimizes the overall obnoxiousness when far from a demand-point, but reflects the local effects when close to a demand-point. Corresponding to this objective we use the non-negative weights w^1. The second objective is the standard formulation for locating an attractive facility by minimizing the weighted sum of the distances (called minisum or median). Please note that we use non-negative weights w^2 with this objective, so that the two objectives may be weighted differently with respect to each of the n demand points. We may also use two different norms, $p_1 \neq p_2$.

If we are modeling where to place a new airport (the example in Section 4), the first weight w_j^1 may depend on the population at demand point j (e.g. city), and the second weight w_j^2 may be the expected number of passengers on a yearly basis from demand point j.

S is the set of feasible solutions. Because of the obnoxious effects from the new semi-obnoxious facility, we assume that it is forbidden to place it too near an existing facility. Therefore, we require, that $\| x - a_j \|_{p_1} > \epsilon, \quad j = 1, \dots, n$, where ϵ is a small positive number. Notice, that this assumption makes the two objective functions Lipschitzian in the feasible set S.

An obvious question for this model would be, if all feasible points are efficient? The answer is that there does exist examples where all feasible points are efficient, but that will probably not be the case in a realistic set-up.

2.1 The idea of the BSSS algorithm

In this paper the idea behind the BSSS method will be applied to the BSPL problem (and also to the BSNL problem). Therefore we briefly review the method below.

Suppose that the feasible region S is contained in a disjoint union of squares of equal size. We put these squares into a list named ES. Next each of these squares are considered separately. Consider one of the squares, say Q_i. We divide Q_i into four sub-squares Q_{i1}, Q_{i12}, Q_{i3} and Q_{i4} of equal size. For each of these sub-squares, say Q_{i1}, lower bounds on the objective function values $(f(x), g(x))$, $x \in Q_{i1}$, are found. By comparing this lower bound with a sample set of objective function values (stored in a list called EFV) it may be determined that square Q_{i1} contains only inefficient points (this is done by the Dominance Check Routine DCR(Q_{i1})). If this is the case square Q_{i1} is called an inefficient square and may be deleted from further consideration. The squares that cannot be classified as inefficient are put into the ES list and will later be divided further into four new sub-squares. The process continues until the side-lengths of all the remaining squares (those that are not classified as inefficient) in ES are below some pre-specified value ϵ. This procedure is justified provided the two objective functions $f(x)$ and $g(x)$ are Lipschitzian in the feasible set S (which is the case in the present paper).

A few comments on the procedure are appropriate. The sample list of objective function values kept in (the sorted) list EFV (Efficient Function Value) are used to dominate sub-squares with poor objective function value bounds. Therefore the values should in a way represent the objectives' behavior over the feasible region. This is done by calculating objective function values in the centers of all the squares, if the center is in the feasible region S, and otherwise in some other feasible point, and then deleting pairs of objective function values being dominated by other objective function values in the EFV list. It is also essential that we use good lower bounds for the objective function values over the squares. If the bounds are poor, the convergence of the algorithm may be slow, because we will end up with a large number of squares. These bounds are explained in detail in Sections 2.2 and 2.3. Finally, we need to check if a square is contained in the feasible region, is overlapping the region or is outside the region. For a discussion of this issue we refer to the paper by Hansen *et al.* [10].

The output from the algorithm is an ordered set of "efficient" squares. By "efficient" we mean that we have not been able to dominate them for this particular choice of ϵ. This is why the procedure is only an approximation. The "efficient" squares can be associated with a certain objective function value, to illustrate the trade-off between the two objectives. This can be done by giving the squares a color corresponding to the value of the first objective. This will illustrate how one objective improves as the other gets worse, and visualize the objective function values being favored in the different "efficient" regions. In the remaining part of this section we assume to have an initial approximation of the feasible region S by equal size squares.

2.2 Calculating lower bounds

In order to calculate lower bounds on the two objectives, we use an approximation of the weighted distances. This distance approximation is illustrated in Figure 1 for the l^2 norm. The lower bound for the distance is found in Hansen *et al.* [10], and the upper bound for the distance is an obvious extension of the same idea, found in Hansen *et al.* [11].

The plane is divided into 9 regions, obtained by extending the four sides of Q_i. The regions are the square Q_i, the four side regions, and the four corner regions. The square Q_i will be in the center.

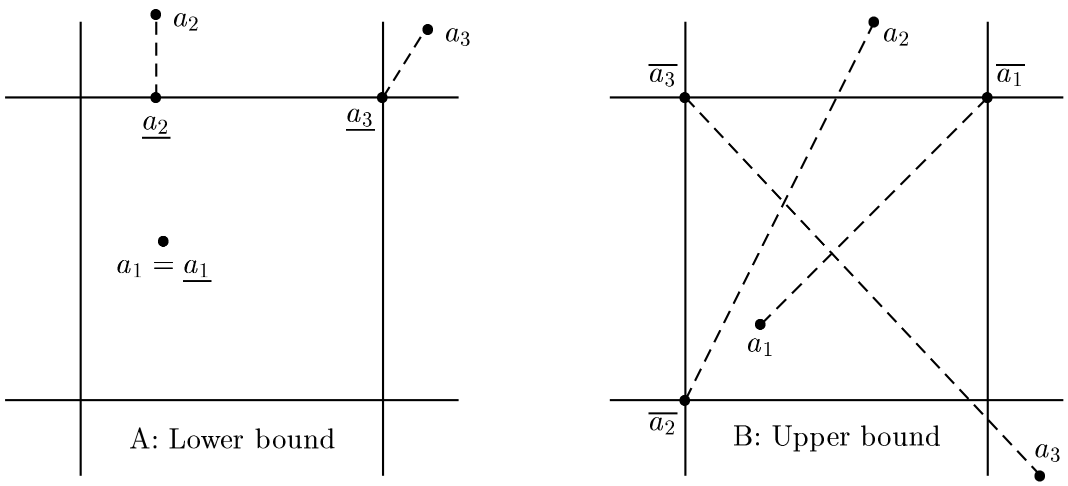

Figure 1: Lower and upper bounds on the distances.

Now let a_j be a particular location. With this location we associate a closest point $\underline{a_j} \in Q_i$ and a furthest point $\overline{a_j} \in Q_i$, see Figure 1. We may then calculate a lower bound on the values of f and g in Q_i as follows:

$$\begin{array}{rcll} \underline{f}(Q_i) & = & \sum_j w_j^1 (\| \overline{a_j} - a_j \|_{p_1})^{-b} & \text{Case B in Figure 1} \\ \underline{g}(Q_i) & = & \sum_j w_j^2 \| \underline{a_j} - a_j \|_{p_2} & \text{Case A in Figure 1} \end{array}$$

Clearly, $(\underline{f}(Q_i), \underline{g}(Q_i) \leq (\min_{x \in Q_i} f(x), \min_{y \in Q_i} g(y))$. Therefore we can use the bound $\underline{z}(Q_i) = (\underline{f}(Q_i), \underline{g}(Q_i))$ for efficiency checking in the algorithm. If we at some point have found a sample value $x \in S$, such that $(f(x), g(x)) < (\underline{f}(Q_i), \underline{g}(Q_i))$, then, clearly all points in Q_i are dominated by x. It follows that square Q_i contains only inefficient points. Therefore it is not necessary to consider Q_i anymore. This bound approach can be used for any $p \in [1; \infty]$. Please note that the bounds obviously converge when the squares get smaller.

2.3 Exact lower bound

Since the minisum objective is a nice convex function, it is possible to calculate an exact lower bound for the squares in most situations. The level sets of a convex function are convex sets, and the gradient can therefore be used as follows.

For a square Q_i with corners c_1, c_2, c_3 and c_4, find the corner c_h with the minimum function value $g(c_h)$. If the direction of steepest descent "**points away**" from the square Q_i, then the lower bound $\underline{g}(Q_i)$ is exactly $g(c_h)$. By "pointing away" we mean that the direction of steepest descent has an angle of at least 90 degrees with the sides of Q_i, see case A in Figure 2. If this angle is less than 90 degrees, the minimum value over Q_i is not in c_h, but on the line segment between c_h and the corner, the direction points out, see case B in Figure 2. Finally, if the direction points **into** Q_i, the minimum value is **not** in c_h but inside Q_i.

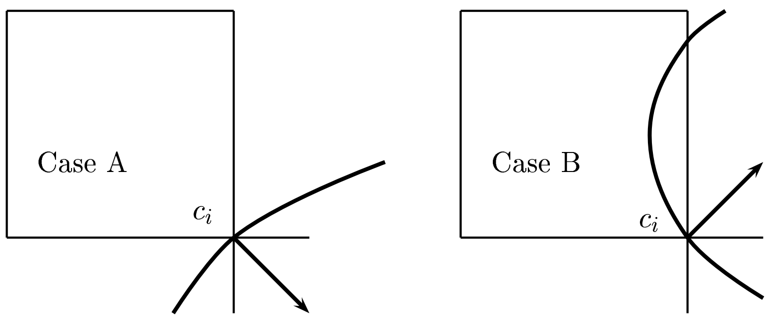

Figure 2: Exact lower bound, depending on directional derivative

From the above, an exact lower bound can easily be computed, if the directional derivative points away from the square. We only need to compute four function values and the directional derivative in the minimum value corner. Case A in Figure 3 will occur in most evaluations, but not in all.

The directional derivative $g'(x_0, y)$ of g at $x_0 \in S$ in the direction y is defined as follows:

$$g'(x_0, y) = \nabla\ g(x_0) \cdot y$$

where $\nabla g(x_0)$ is the gradient of g evaluated in x_0.

If we consider the l^2 norm, the gradient looks as follows:

$$\nabla\ g(x_0) = \left(\sum_j \frac{w_j^2(x_{01} - a_{j1})}{\| x_0 - a_j \|}, \sum_j \frac{w_j^2(x_{02} - a_{j2})}{\| x_0 - a_j \|} \right)$$

This reveals the well-known problem; if x_0 is at a demand point, the gradient is undefined because of the denominator being zero. In this case we also have to use the lower bound of Section 2.2.

Expressions for the gradient for general l^p norms, can be derived for any $p \in [1; \infty]$. Actually, the only assumption needed for the exact lower bound to be valid, is that the level sets are convex. The reason for deriving tighter bounds is to speed up convergence of the algorithm.

2.4 BSSS algorithm for the BSPL problem

Notation:

Q_i	Square number i
$\underline{z}(Q_i) = (\underline{f}(Q_i), \underline{g}(Q_i))$	Lower bounds for Q_i.
ES	List of Efficient Squares. Note that this is only a name for squares that have not been proven inefficient.
ECL	Efficient Candidate List (of squares of equal size). It consists of the four sub squares of all the squares in ES.
EFV	List of Efficient Function Values. Function values are calculated at different points in the feasible region, and the nondominated ones (at this time in the routine) are in this list.
DCR(Q_i)	Dominance Check Routine for Q_i (with EFV). Is briefly explained in Section 2.1.

The idea for the DCR routine was found in [3], and earlier used by the authors in [13].

Planar Algorithm:

1. Initialize

   Find an equal size square approximation $Q_1, Q_2, \ldots, Q_N$ of S

Put Q_i in ES $\forall i = 1, 2, \ldots, N$.

Let L be the length of a side of Q_1

Define the tolerance level ϵ

2. Creating New Squares

For each $Q_i \in$ ES do

Create 4 sub-squares $Q_j, j = 1, 2, 3, 4$, put the Q_j's in ECL and delete Q_i from ES

Set $L = \frac{L}{2}$

3. Efficiency Update

Update EFV by calculating some function values from the Q_j's

For each $Q_j \in$ ECL do

Calculate $\underline{z}(Q_j) = (\underline{f}(Q_j), \underline{g}(Q_j))$ using exact lower bounds when possible

Make DCR(Q_j) with EFV

If Q_j is efficient compared with EFV then add Q_j to ES

4. Termination Test

If $L < \epsilon$ Terminate with ES as the solution list

Else go to Step 2

3 The network case : The BSNL problem

In this section we adapt the BSSS method to the network case. However, instead of dividing big squares into smaller squares, we divide edges into sub-edges. This will be explained in detail in Section 3.1. Assume we have an undirected connected network $G(\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ where $|\mathcal{V}| = n$ nodes, and a finite set of edges (arcs) $\mathcal{E} = \{(v_i, v_j), (v_k, v_l), \ldots, (v_p, v_q)\}$ with $|\mathcal{E}| = m$. Edges may also be denoted by e. All edges have a strictly positive length. Each node v_j carries two non-negative weights (w_j^1, w_j^2), one for the obnoxious criterion and one for the desirable criterion.

The model is the same as (2), except that the set of possible new locations is the entire network. With our choice of obnoxious objective function, however, x cannot be located in a node. The BSNL problem is then:

$$\begin{array}{ll} \min & f(x) = \sum\limits_j w_j^1 \ (d(x,v_j))^{-b}, \qquad b > 0 \\ \min & g(x) = \sum\limits_j w_j^2 \, d(x,v_j) \\ \text{s.t.} & \\ & x \in G(\mathcal{V},\mathcal{E}) \end{array} \tag{3}$$

where $d(x, v_j)$ is the shortest distance from point x to node v_j. The authors are well aware that the obnoxious objective function is not as appropriate on the network model, as in the planar model, but we have decided to use it for comparison purposes, see Section 4. The solution procedure is described shortly in Section 3.1 and the algorithm is presented in Section 3.4. The approximation algorithm is a very general and intuitive approach and can be used for complicated objective functions.

3.1 The Edge Dividing algorithm

The idea of the Edge Dividing (ED) algorithm is similar to the idea behind the BSSS algorithm. First we divide each edge into two sub-edges. Then bounds on the objective function values on each sub-edge are calculated. Furthermore, a sample set of objective function values are calculated. If the bounds calculated for a sub-edge are dominated by one (or more) of the sample set objective function values then the sub-edge is dominated and may be deleted from further consideration.

The bounds are derived in detail in Sections 3.2 and 3.3. The sample set of objective function values are calculated in the middle (center) of the sub-edges. Nondominated criterion values are kept in the EFV list. Please note that only an approximation of the efficient set is found.

The output from the algorithm is an ordered set of "efficient" sub-edges. This general procedure, however, has a few disadvantages. The efficient set (or part of it) may be an edge-segment. This sub-edge will obviously remain in the ES list, but the sub-edge will be divided into sub-edges again and again. This reveals that the ES set will probably almost double in size, when we half the ϵ value. This can in fact be used as an alternative stopping criterion.

3.2 Calculating lower bounds

We need both upper and lower bounds on the distance $d(x, v_j)$, where x can be any point on the edge (or sub-edge) e_i. We refer to the lower bound of this distance by $\underline{d(e_i, v_j)}$ and to the upper bound by $\overline{d(e_i, v_j)}$. Assume $e_i \in (v_h, v_k)$, and x_h is the endpoint of e_i closest to v_h, and that x_k is the endpoint of e_i closest to v_k.

The upper-bound may be calculated as follows:

$$\overline{d(e_i, v_j)} = \min\{d(v_j, v_h) + d(v_h, x_h), d(v_j, v_k) + d(v_k, x_k)\} + d(x_h, x_k)$$

and the lower-bound may be calculated as follows:

$$\underline{d(e_i, v_j)} = \min\{d(v_j, v_h) + d(v_h, x_h), d(v_j, v_k) + d(v_k, x_k)\}$$

These two bounds can be easily calculated as illustrated in Figure 3, whenever the distance matrix D, of shortest distances between all pairs of nodes, is available.

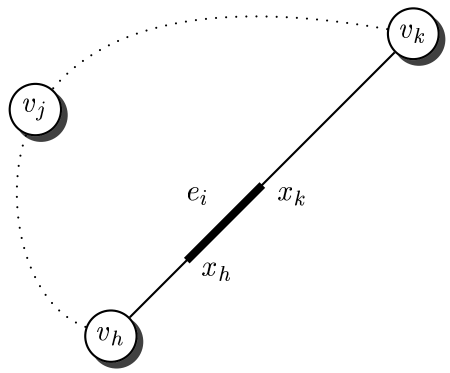

Figure 3: Calculating distance bounds.

Using these bounds we can calculate the lower bounds on the objective function values as

$$\begin{aligned} \underline{f}(e_i) &= \sum_j w_j^1 \left(\overline{d(e_i, v_j)}\right)^{-b} \\ \underline{g}(e_i) &= \sum_j w_j^2 \, \underline{d(e_i, v_j)} \end{aligned}$$

3.3 Exact bounds

In this section we derive some exact bounds, specifically for our choice of objective functions.

The distance function $d(x, v_j)$ is a concave functions on an edge (subedge). Therefore, $g(x)$ is a concave function on an edge, and the minimum is always in one of the (sub-edge) endpoints. So we have an exact lower bound as follows.

$$\underline{g}(e_i) = \min\{g(x_h), g(x_k)\} \tag{4}$$

Now lets consider $f(x)$. Since $d(x, v_j)$ is both positive and concave, $(d(x, v_j))^{-b}$ is convex. Therefore, $f(x)$ is convex on an edge. If we are looking at the sub-edge from x_h to x_k as illustrated in Figure 3, and the derivatives at the endpoints have the same sign, then an exact lower bound is simply the smallest endpoint value. That is, if

$$sign\left(\frac{\partial^+}{\partial x_{(v_h,v_k)}} f(x_h)\right) = sign\left(\frac{\partial^+}{\partial x_{(v_h,v_k)}} f(x_k)\right) \tag{5}$$

then

$$\underline{f}(e_i) = \min\{f(x_h), f(x_k)\} \tag{6}$$

where $\frac{\partial^+}{\partial x_{(v_i,v_j)}} f(x_k)$ denotes the derivative in the direction from v_i towards v_j, and we want to know if the function increases or decreases. The "+" indicates right derivative, so even in a break-point this derivative is well-defined. If (5) does not hold, the bound in Section 3.2 has to be applied. For more general objective functions, the bounds in Section 3.2 may be needed more often.

3.4 ED algorithm for the BSNL problem

Notation:

e_i	Sub-edge number i
$\underline{z}(e_i) = (\underline{f}(e_i), \underline{g}(e_i))$	Lower bounds for e_i.
ES	List of Efficient Sub-edge. Note that this is only a name for sub-edges that have not been proven inefficient.
ECL	Efficient Candidate List (of sub-edges). It consists of the two sub-edges of all the sub-edges in ES.
EFV	List of Efficient Function Values. Function values are calculated at different points on the network, and the nondominated ones (at this time in the routine) are in this list.
DCR(e_i)	Dominance Check Routine for e_i (with EFV).
L	Length of a longest edge in ES.

Network Algorithm:

1. Initialize

   Find the shortest path matrix D.

   Put all edges $e_1, e_2, \ldots, e_m$ in ECL.

   Let L be the length of a longest edge in ECL.

   Define the tolerance level ϵ.

   Calculate criterion values in all midpoints to make an initial EFV list.

2. Efficiency Update

   For each $e_i \in$ ECL do

Calculate $\underline{z}(e_i) = (\underline{f}(e_i), \underline{g}(e_i))$ using (4) and the exact bound (6) if possible

Make DCR(e_i) with EFV

If e_i is Efficient compared with EFV then add e_i to ES

Update L (as a longest sub-edge in ES)

3. Termination Test

If $L < \epsilon$ Terminate with ES as the solution list

4. Creating New Sub-edges

For each $e_i \in$ ES do

Split e_i into two sub-edges e_{i1} and e_{i2} of equal length.

Add e_{i1} and e_{i2} to ECL and delete e_i from ES

Update EFV by calculating criterion values on the middle of all sub-edges e_j in ECL

Go to Step 2

4 An airport example

To illustrate the usefulness of the two models we present an application. Currently, there is a debate in Denmark as to the location of a new international airport in the mainland Jutland in order to replace an existing one. The existing airport is located near a small city called Tirstrup approximately 45 km to the North-East of Århus, the largest city in Jutland (with about 215.000 inhabitants). The existing airport is located in an area where not many people are living and where not many companies are based. Furthermore, the infrastructure of this area is not too good. For example it takes about 1 hour to go from Århus to Tirstrup. Many companies (and people) think that this is too much time to spend on transportation to the airport.

It is believed that a new international airport located not too far away from Århus would be attractive to a lot of companies (and people). However, customers (companies/people) living nearby Århus are more likely to use the new airport than customers living far away from Århus. Therefore, we will consider only a region of potential locations with x-coordinates between 60 and 140, and y-coordinates between 100 and 180, see Figure 4. Furthermore, we have divided Jutland into three zones, namely a 100% zone, a 50% zone, and a 20% zone, see Figure 4. The weighting zones should reflect the fact that customers far away from the chosen region will use the new airport less frequently than customers close by or within this region. These three weighting zones will be used when defining the transportation objectives later in this section. We have chosen 42 cities to represent the customers in Jutland, ranging in population from 2574 (Hanstholm) to 215587 (Århus) inhabitants as demand points. Distance is measured in kilometers, and the ϵ-value used is 0.15 km (150 meters) for both the planar case and the network case. Origo is placed on the German island of Sylt.

Next, let us describe the parameters for the two objective functions. For the planar model we have used the Euclidean distance and a b-value of two. For the network model, the distance is always the shortest distance in the network. The b-value is two. The edge lengths are road distances collected from an intercity distance table. All input data is available from the corresponding author.

For the obnoxious criterion we have used weights w_j^1 ="population in city j". This is a simple form of letting the larger cities count more than the smaller cities. For the transportation cost objective we have used weights w_j^2 ="population in city j multiplied by the weight of the zone in which the city is located". This means that cities nearby Århus count much more than cities far away from Århus, reflecting the fact that customers far away from Århus are likely to use the new airport less frequently than customers living nearby Århus.

Whether the city population is an appropriate measure of passengers is not an issue here. The data for the example is presented in Table 1, and it is used for both the planar and the network problem. The three dummy-nodes in Table 1 are introduced only to make the road-network in Figure 7 more realistic, and are located right to the west of Århus. These nodes are introduced because the main highway follows a half-circle around Århus.

First we present the results of the planar model. The norm to the negative power function is illustrated in Figure 5, covering the region of $[60,140] \times [100,180]$. The peaks indicate the cities, with function values going to infinity. As can be seen from Figure 5 it may be hard to find an exact lower bound for this function. The minisum global optimum is attained in $(110,145)$ with a value of $3,27 \cdot 10^7$. The minisum function is not plotted since it is just a convex function.

The efficient region is illustrated on the map in Figure 6. For clarity, we have drawn two minisum level curves. The inner level curve is minisum values 10% above the global minisum minimum $(3,6 \cdot 10^7)$, and the outer level curve is 20% above $(3,92 \cdot 10^7)$.

Figure 6 reveals three efficient regions. The central region just west-north-west of Århus containing the global minisum minimum, and with minisum-values within 10% of the minimum. The central region reflects in which direction the obnoxious objective

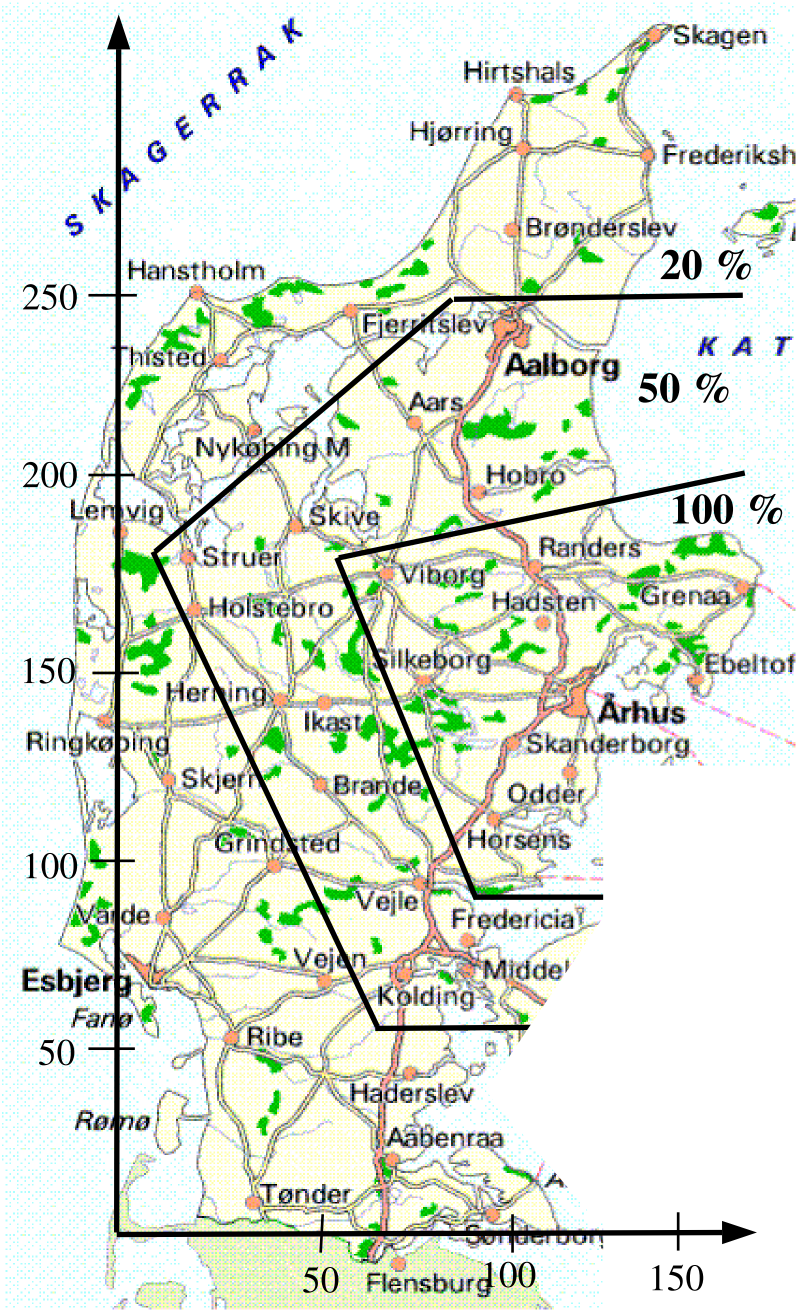

Figure 4: Jutland divided into three weighting-zones. Coordinates are in kilometers.

City (j)	a_{j1}	a_{j2}	w_j^1	w_j^2
Esbjerg	7.17	69.31	73422	14684.4
Tønder	34.416	8.126	8161	1632.2
Ribe	28.202	52.102	8046	1609.2
Kolding	72.178	68.832	53012	26506
Vejle	76.002	93.21	47839	23919.5
Horsens	95.122	110.418	48410	48410
Skanderborg	99.902	130.494	12067	12067
Århus	118.066	142.922	215587	215587
Randers	106.116	177.338	56123	56123
Viborg	67.876	175.904	31872	31872
Silkeborg	76.958	146.746	36762	36762
Ikast	52.102	141.01	14014	7007
Herning	40.63	141.966	29231	14615.5
Holstebro	18.642	166.344	30770	15385
Struer	17.208	181.206	11272	5636
Skive	44.454	188.332	20557	10278.5
Hadsten	108.028	162.042	6616	6616
Grenå	158.696	172.08	14441	14441
Hobro	91.298	196.936	10704	5352
Aars	74.568	216.056	7066	3533
Ålborg	98.468	240.434	119157	59578.5
Fredericia	88.43	78.392	29376	14688
Haderslev	74.09	42.064	21106	4221.2
Aabenrå	69.31	20.076	16218	3243.6
Vejen	52.58	66.442	8507	1701.4
Brønderslev	99.902	267.202	11365	2273
Hjørring	103.248	289.668	24889	4977.8
Frederikshavn	135.274	288.234	24768	4953.6
Bjerringbro	83.547	169.246	7201	7201
Varde	10.994	83.172	12478	2495.6
Grindsted	38.718	97.034	9497	1899.4
Skjern	11.95	119.978	6949	1389.8
Ringkøbing	-4.302	135.274	9166	1833.2
Brande	50.668	119.022	6214	3107
Lemvig	0	185.464	7302	1460.4
Nykøbing	33.46	212.71	9319	1863.8
Thisted	25.334	231.352	12609	2521.8
Hanstholm	19.12	249.516	2574	514.8
Fjerritslev	58.316	244.736	3332	666.4
Hirtshals	100.858	301.14	6949	1389.8
Skagen	136.708	315.958	10674	2134.8
Ebeltoft	146.746	144.834	4396	4396
Dummi North	113	152	0	0
Dummi West	109	144	0	0
Dummi South	109	137	0	0

Table 1: Locations $a_j = (a_{j1}, a_{j2})$ and weights (w_j^1, w_j^2) of 42 cities in Jutland.

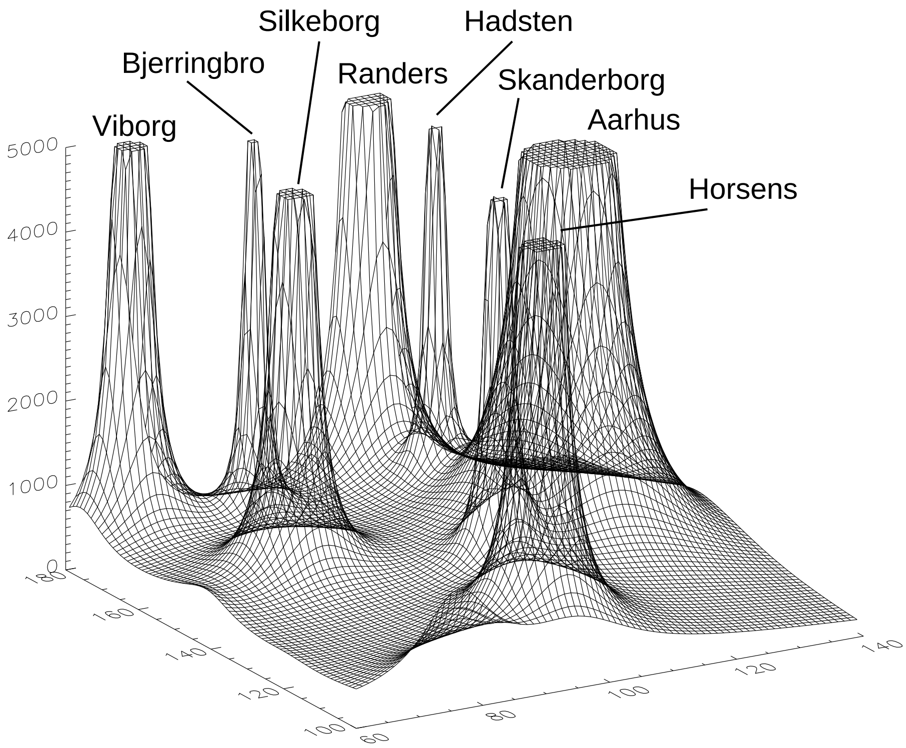

Figure 5: Surface-plot of the obnoxious objective-function.

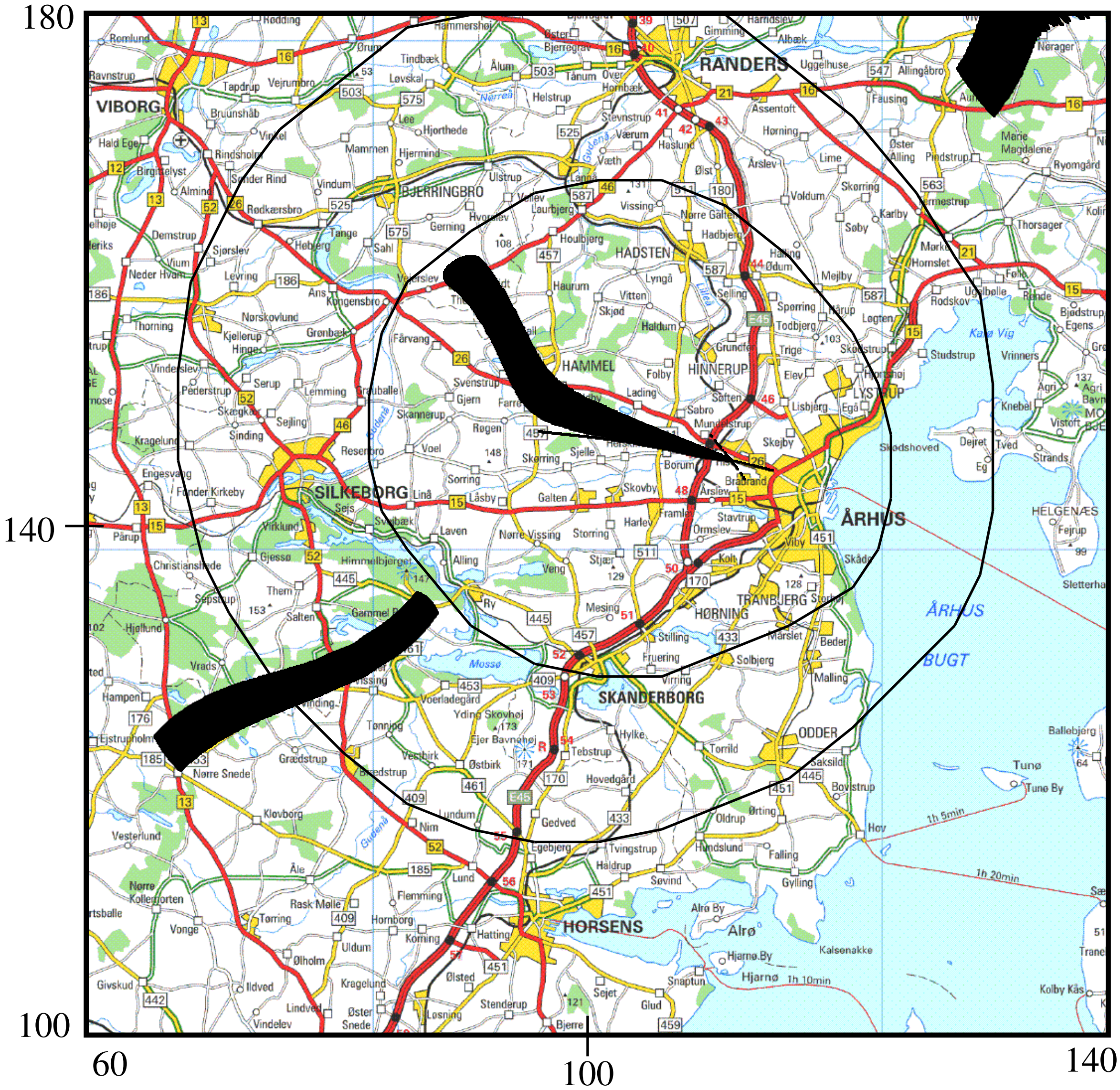

Figure 6: Efficient regions for airport location

Region	f-value	g-value $(\cdot 10^7)$
A	$]\infty$; 3772]	[3.63 ; 3.80]
B	[3756 ; 2244]	[3.86 ; 3.92]
C	[713 ; 570]	[4.29 ; 4.45]
D	[568 ; 419]	[4.49 ; 4.70]
E	[419 ; 316]	[4.77 ; 4.93]
F	[316 ; 222]	[5.18 ; 5.71]
G	[222 ; 193]	[5.91 ; 6.34]

Table 2: Objective function values for the regions indicated in Figure 7.

decreases, namely north-west. This region has the highest obnoxious values, ranging from 3400 (at the minisum optimum) to 675 in the north-western part of the region. The south-west region has minisum values from approximately 10% to 25% above the minimum. This region has obnoxious values from 675 to 440. The last efficient region is the north-east region with minisum values more the 25% above the minimum. This region has the lowest obnoxious values, below 440, simply because there are no major cities in this part of the country as can be seen in Figure 4. As a matter of fact the existing airport at Tirstrup is nearby this region. Potential locations for a new airport should be found within these efficient regions.

Next we present the results of the network model. For this model only the resulting network, with the efficient sub-edges, is presented, see Figure 7. The minisum objective function has its global minimum in the node representing the city of Århus. The obnoxious objective function has its global minimum outside the target region.

In Figure 7 the seven efficient regions are indicated by the letters A, B, ..., G, and the corresponding objective function value intervals are presented in Table 2. The region A around Århus has the lowest transportation costs, but also quite high obnoxious values. The most deserted subedge, region G, has the lowest obnoxious values, but almost two times the lowest transportation cost. The trade-off between the two objectives is well represented by Table 2. Figure 7 reveals that a possible location of the new airport could be in an area north-west of Århus.

5 Concluding remarks

In this paper we have set up two bicriterion location models for locating one obnoxious facility, namely one for the planar case and one for the network case. Efficient (well-working) solution algorithms based on the well-known BSSS algorithm has been proposed.

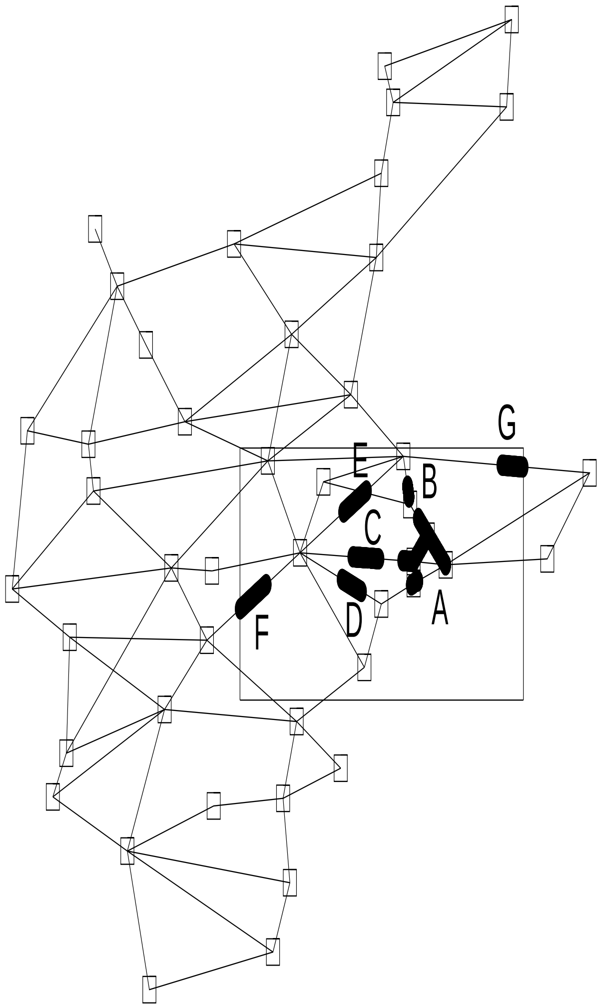

Figure 7: Road-network of Jutland. Bold parts constitute the efficient set.

Both models are easily extended to multiple criteria. All that needs to be changed is the DCR operation.

Even though the planar and the network model may seem distinct in structure, they are designed to solve the same real-life problem. Often a combination of the two models would be preferable. For example, modeling air pollution such as noise makes most sense in the planar model, whereas the network model would be the correct description of a road network with distances or travel times as coefficients. One possible combination is to embed the network on top of the plane, so that each point on the network corresponds to a point in the plane, but not necessarily the other way around.

Another issue is the choice of obnoxious criterion functions. We have used the negative power function also used in Brimberg and Juel [1]. Of course, many other functions may be used, and for more complicated functions, the approximation approach described in this paper may be the only applicable approach.

It may also be appropriate to have weights depending on distance. However, in most exact models this will cause mathematical difficulties. In the airport example presented in Section 4, the number of yearly passengers from a city using the new airport, most probably depends negatively on the distance.

It should also be considered what kind of pull objective (cost function) is appropriate. We have only considered the minisum. It should also be noted that for some objectives an exact bound, or at least an improved bound, may be applied.

The output of the models reveal the trade-off between the two negatively correlated criteria. We conclude that the two proposed models are good tools for obnoxious location decisions. Finally, we have illustrated the models on a real-life application.

References

[1] J. Brimberg and H. Juel. A bicriteria model for locating a semi-desirable facility in the plane. *European Journal of Operational Research*, 106:144–151, 1998.

[2] J. Brimberg and H. Juel. On locating a semi-desirable facility on the continuous plane. *International Transactions in Operational Research*, 5:59–66, 1998.

[3] J. Brumbaugh-Smith and D. Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43:216–224, 1989.

[4] E. Carrizosa, E. Conde, and D. Romero-Morales. Location of a semiobnoxious facility. A biobjective approach. In 1996 Torremolinos, editor, *Advances in multiple objective and goal programming*, pages 338–346. Springer-Verlag, Berlin-Heidelberg, 1997.

[5] E. Carrizosa and F. Plastria. Location of semi-obnoxious facilities. *Studies in Locational Analysis*, 12:1–27, 1999.

[6] M. Ehrgott. *Multicriteria Optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems*. Springer, 2000.

[7] E. Erkut and S. Neuman. Analytical models for locating undesirable facilities. *European Journal of Operational Research*, 40:275–291, 1989.

[8] R.L. Francis, L.F. McGinnis, and J.A. White. *Facility layout and location: An analytical approach*. Prentice Hall, New Jersey, 1992.

[9] H.W. Hamacher, M. Labbé, S. Nickel, and A.J.V. Skriver. Multicriteria semi-obnoxious network location problems (MSNLP) with sum and center objectives. *Working paper*, 2000-6, 2000. Department of Operations Research, University of Aarhus, Denmark.

[10] P. Hansen, D. Peeters, D. Richard, and J.F. Thisse. The minisum and minimax location problems revisited. *Operations Research*, 33:1251–1265, 1985.

[11] P. Hansen, D. Peeters, and J.F. Thisse. On the location of an obnoxious facility. *Sistemi Urbani*, 3:299–317, 1981.

[12] R.F. Love, J.G. Morris, and G.O. Wesolowsky. *Facilities Location : Models & Methods*. North-Holland, New York, 1988.

[13] A.J.V. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest path problems. *Computers and Operations Research*, 27:507–524, 2000.

[14] R.E. Steuer. *Multiple criteria optimization: Theory, Computation, and Application*. Wiley, New York, 1986.

Multicriteria Semi-obnoxious Network Location (MSNL) Problems with Sum and Center Objectives

Horst W. Hamacher
Fachbereich Mathematik
Universität Kaiserslautern
Kurt-Schumacher-Strasse 26
67663 Kaiserslautern
Germany

Martine Labbé
Service de Mathématiques de la Gestion
Université Libré de Bruxelles
Brussels
Belgium

Stefan Nickel
ITWM
Universität Kaiserslautern
Ottlieb-Daimler-Strasse
Gebäude 49
67663 Kaiserslautern
Germany

Anders J. V. Skriver*
Department of Operations Research
University of Aarhus
Building 530, Ny Munkegade
DK - 8000 Århus C
Denmark

August 16, 2001

Abstract

Locating a facility is often modeled as either the maxisum or the minisum problem, reflecting whether the facility is undesirable (obnoxious) or desirable. But many facilities are both desirable and undesirable at the same time, e.g. an airport. This can be modeled as a multicriteria network location problem, where some of the sum-objectives are maximized (push effect) and some of the sum-objectives are minimized (pull effect).

We present a polynomial time algorithm for this model along with some basic theoretical results, and generalize the results also to incorporate maximin and minimax objectives. In fact, the method works for any piecewise linear objective functions. Finally, we present some computational results.

Keywords: MCDM, Multicriteria, Obnoxious, Semi-obnoxious, Facility Location, Networks.

1 Introduction

There are a number of models that deal with the problem of locating (placing) a new facility on a network. Most of these models locate a desirable facility, such as a supermarket or a

*Corresponding author.

fire station, where the objective is to keep the new facility close to its users (pull effect). There are also some models describing how to locate an obnoxious (undesirable) facility such as a nuclear power plant or a dump site which the users want to locate far away (push effect). Many facilities can, however, be thought of as semi-obnoxious. Such facilities could be airports, train stations or other noisy service facilities. It could also be the above-mentioned dump site that, with respect to transportation costs, should be located centrally, but, in the opinion of the citizens, should be located distantly. These location problems could with obvious advantages be formulated as multicriteria network location problems. In this way the trade-off between the different objectives can be revealed, making a good basis for an overall decision. Different aspects of the problem can be described by different objectives. Such objectives could be transportation costs, travel time, air pollution or minimizing the number of citizens within a certain radius of the facility. Another situation arises when we have more decision makers, each having their own objective function. When we solve a problem with more than one objective, it is highly unlikely that one solution is optimal for all objectives. Instead, the solution is the set of efficient or Pareto locations, i.e. solutions where we cannot improve any objective without at least one other objective being worsened.

Bicriterion models for the planar case of the problem is presented in Brimberg and Juel [1], Carrizosa *et al.* [2] and Andersen and Skriver [10]. In Andersen and Skriver [10] an approximation solution method for the bicriterion network location problem is also presented. A general solution method for the multicriterion median-problem is presented in Hamacher *et al.* [5].

As one notices, the terminology for location problems is not unique. Therefore we introduce in the following a classification scheme for location problems that should help get an overview over the manifold area of location problems.
We use a scheme which is analogous to the one introduced successfully in scheduling theory. The presented scheme for location problems was developed in Hamacher and Nickel [6] and Hamacher *et al.* [5].
We have the following five position classification

$$pos1/pos2/pos3/pos4/pos5 \ ,$$

where the meaning of each position is explained in Table 1:
If we do not make any special assumptions in a position, we indicate this by a $\bullet$.

The rest of the paper is organized as follows. In Section 2 we give some definitions and

Position	Meaning	Usage (Examples)
1	number of new facilities	
2	type of problem	**P** planar location problem **D** discrete location problem **G** network location problem
3	special assumptions and restrictions	$w_m = 1$ all weights are equal $\mathcal{R}$ a forbidden region
4	type of distance function	l_1 Manhattan metric $d(\mathcal{V}, \mathcal{V})$ node to node distance $d(\mathcal{V}, G)$ node to point distance
5	type of objective function	$\sum$ median problem $\sum_{obnox}$ anti-median problem max center problem $\max_{obnox}$ anti-center problem

Table 1: Classification scheme for location problems.

describe the problem. The general solution procedure is described in Section 3, and in Section 4 we present a different approach that works only in the bicriteria case. In Section 5 we discuss how the general solution procedure can also be used with center objectives. Computational results are presented in Section 6, and we conclude the paper in Section 7.

2 Problem formulation and definitions

We are given a (strongly) connected network $G(\mathcal{V}, \mathcal{E})$ with nodeset $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ where $|\mathcal{V}| = n$ nodes, and edgeset $\mathcal{E} = \{(v_i, v_j), (v_k, v_l), \dots, (v_p, v_q)\}$ with $|\mathcal{E}| = m$ edges. If the underlying graph is directed it is denoted G_D, and the edge $e = (v_i, v_j)$ has head v_j and tail v_i. If the underlying graph is undirected, it is just denoted G, and $e = (v_i, v_j) = (v_j, v_i)\ \forall e \in \mathcal{E}$. We define the set of objectives as $\mathcal{Q} = \{1, 2, \dots, Q\}$. Each node v_i carries Q weights $(w_i^1, w_i^2, \dots, w_i^Q)^t$, where $w_i^q > 0, \forall q \in \mathcal{Q}$, so we may refer to the matrix of weights by $W_{Q \times n}$. Each edge $e \in \mathcal{E}$ has length $l(e) \in R_+$.

By $d(v_h, v_k)$ we denote the distance between v_h and v_k, is given by the length of a shortest path between v_h and v_k. A point $x \in G(\mathcal{V}, \mathcal{E})$ can be located both at a node or on an edge. This is often referred to as absolute location.

We define a **point** x on a directed edge $e = (v_i, v_j)$ as a tuple $x = (e, t), t \in [0, 1]$, with

$$d(v_k, x) = d(v_k, v_i) + tl(e) \quad \text{and} \quad d(x, v_k) = (1 - t)l(e) + d(v_j, v_k)$$

for any $v_k \in \mathcal{V}$. A **point** x on an undirected edge $e = (v_i, v_j)$ is defined as a touple

$x = (e, t), t \in [0, 1]$, with

$$d(x, v_k) = \min\{d(v_k, v_i) + tl(e), d(v_k, v_j) + (1 - t)l(e)\}$$

for any $v_k \in \mathcal{V}$. Notice that $d(v_i, x) = tl(e)$ and $d(x, v_j) = (1 - t)l(e)$ for $x = (e, t)$. Since $v_i = (e, 0)$ and $v_j = (e, 1)$, all nodes of the network are also points of the network.

The set $\{(e, t) | t \in (t_1, t_2), t_1, t_2 \in [0, 1]\}$, forming an open **subedge** on e, is denoted $(e, (t_1, t_2))$ for any $e \in \mathcal{E}$. Of course this set is empty, unless $t_2 > t_1$. Similarly, we define closed and half right/left open subedges.

We formulate the model with the maxisum and minisum objectives, which are obviously negatively correlated. These objective functions are often referred to as the **weighted anti-median** and **median** of a network. In Section 5 we discuss the maximin and minimax objectives. For the undirected problem the objective functions are defined by

$$f^q(x) = \sum_{i=1}^{n} w_i^q \, d(x, v_i) \quad q \in \mathcal{Q} \tag{1}$$

and for the directed case they are defined by

$$f^q(x) = \sum_{i=1}^{n} w_i^q \, (d(x, v_i) + d(v_i, x)) \quad q \in \mathcal{Q} \tag{2}$$

In (2) observe that we for each node v_i make a round-trip from x to v_i and back to x. In some applications it may be more appropriate to look only at the distances out of x or into x. The general undirected problem $1/G/\bullet/d(\mathcal{V}, G)/(Q_1\text{-}\sum_{obnox}, Q_2\text{-}\sum)_{Par}$ is formulated as follows:

$$\begin{array}{lll} \max & f^q(x) & q \in \mathcal{Q}_1 \\ \min & f^q(x) & q \in \mathcal{Q}_2 \\ \text{s.t.} & & \\ & x \in G(\mathcal{V}, \mathcal{E}) & \end{array} \tag{3}$$

$\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$, where $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$. $\mathcal{Q}_1$ is the set of obnoxious objective functions, and $\mathcal{Q}_2$ is the set of desirable objective functions. At most one of the sets are allowed to be empty. If $\mathcal{Q}_1 = \emptyset$ we have the situation discussed in Hamacher, Labbé and Nickel [5]. $f(x) = (f^1(x), f^2(x), \ldots, f^Q(x))^t$.

For simplicity in the succeeding argumentation we multiply all objective functions in $\mathcal{Q}_1$ by -1 in order to minimize instead of maximize. Thus, in the remaining part of the paper we assume that $w_i^q < 0, \forall i = 1, 2, \ldots, n$ and $q \in \mathcal{Q}_1$, and $w_i^q > 0, \forall i = 1, 2, \ldots, n$ and $q \in \mathcal{Q}_2$. We now have a multicriteria minimization model:

$$\begin{array}{lll} \min & f^q(x) & q \in \mathcal{Q}_1 \\ \min & f^q(x) & q \in \mathcal{Q}_2 \\ \text{s.t.} & & \\ & x \in G(\mathcal{V}, \mathcal{E}) & \end{array} \tag{4}$$

In order to find the shortest distances between x and all the nodes, we need the **distance matrix** D of shortest distances between all pairs of nodes. Note that $D_{ij} = d(v_i, v_j)$. This matrix can be calculated in $O(n^3)$ running time using Floyd's algorithm or by applying Dijkstra's algorithm to all n nodes. For details on these graph procedures, see Thulasiraman and Swamy [13]. For an undirected network the distance matrix D is symmetric.

This model is a combination of two well-known models. The minisum and the maxisum models. The solution procedures for these two models are similar, but we will explain the most important details here. For the maxisum problem, some interesting theory is found in Church and Garfinkel [3]. They introduce the concept of bottleneck points, and refer to nodes with degree one as **dangling nodes** (often called pendant nodes). The minisum problem has been well studied, and we refer to Daskin [4] for details.

We will now outline the concept of bottleneck-points as it is presented in Church and Garfinkel [3]. There are two types of bottleneck-points. The edge-bottleneck-points are defined as follows, for each edge $(v_i, v_j) \in \mathcal{E}$: Let x be on the edge (v_i, v_j). If there exists a node $v_k \neq v_i, v_j$ such that

$$D_{ki} + d(x, v_i) = D_{kj} + d(x, v_j)$$

then x is an **edge-bottleneck-point**. It is easily seen, that edge (v_i, v_j) contains an edge-bottleneck-point with respect to node v_k if and only if

$$|D_{ki} - D_{kj}| < l((v_i, v_j))$$

This sets the upper bound for the number of edge-bottleneck-points on an edge to $n - 2$. Now we define the node-bottleneck-points. Assume there exists distinct nodes v_i,v_h and v_k. If there exists a node $v_j \neq v_i, v_h, v_k$ such that

$$D_{ik} + D_{kj} = D_{ih} + D_{hj}$$

then node v_j is a **node-bottleneck-point** with respect to node v_i (and v_i to v_j). Considering the whole edge (v_i, v_j) including the nodes, it contains at most n bottleneck-points. Since there are m edges in G, the total number of bottleneck-points is bounded by mn. It is important to note that the bottleneck-points are independent of the weights. They only depend on the network structure including the edge-lengths. We will denote the **edge-bottleneck-point matrix** of shortest distances from all edge-bottleneck-points to all nodes by B. So B_{ij} is the shortest distance from edge-bottleneck-point B_i to node v_j. This matrix is needed for easy calculation of the objective-values in the bottleneck-points.

When we know the shortest distance matrix D, the bottleneck-points can be calculated in $O(mn)$ running time, because for each edge we have to evaluate all nodes. This can be improved to an algorithm that takes $O(n \log n)$ time, see Hansen *et al.* [7].

In Church and Garfinkel [3] it is shown that there exists a point x, that is either a bottleneck-point or a dangling node that solves the maxisum problem. This is true because the weighted-sum objective is a piecewise linear, **concave** function on the edges, with break-points only in the edge-bottleneck-points. This corresponds to minimizing the weighted sum where all weights are negative. The objective function is then a piecewise linear, **convex** function with break-points only in the edge-bottleneck-points, see f^1 in Figure 1. Note that the optimum need not be unique, it can be a subedge between two (or more) bottleneck-points, or the optimum value may also be obtained on a different edge. It is well-known that the optimum for the minisum problem is found in a node (f^2 in Figure 1). The standard way of solving this problem is to sum the rows of the distance matrix D multiplied by the weights. The row with the smallest weighted sum corresponds to the minisum optimum node. For further details see Daskin [4].

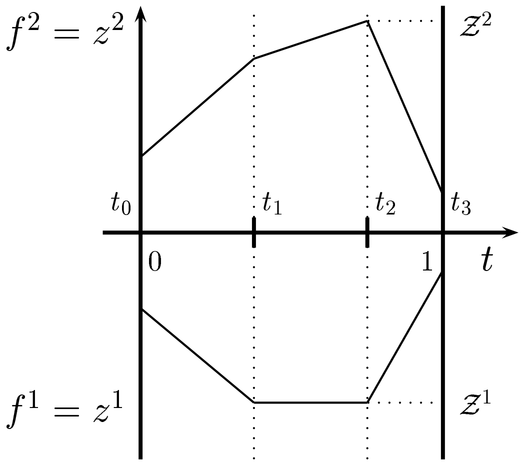

Figure 1: Illustration of the objective functions on an edge.

We denote the set of optimal solutions to a single-objective problem by $\mathcal{X}^q$. The corresponding objective values are denoted by $\mathcal{Z}^q$. Note that these sets of objective-values only contain one value, namely the optimal value, but the notation generalizes to the nondominated set $\mathcal{Z}_{Par}$ defined below.

Solving the Q-criteria semi-obnoxious network location problem means finding the set of efficient points. For an introduction to multiple criteria analysis see Steuer [12].
The definition of efficiency is as follows.

Definition 1 *A solution $x \in G(\mathcal{V},\mathcal{E})$ to (4) is **efficient** (Pareto optimal) iff there does not exist another solution $\bar{x} \in G(\mathcal{V},\mathcal{E})$ to (4) such that $f^q(\bar{x}) \leq f^q(x)$ $\forall q \in \mathcal{Q}$ and $\exists q \in \mathcal{Q}$ s.t. $f^q(\bar{x}) < f^q(x)$. Otherwise x is **inefficient**.*

The set of all efficient/Pareto optimal solutions are denoted by $\mathcal{X}_{Par}$. Efficiency is defined in the decision space. There is a natural counterpart in the criterion space. The criterion space is denoted by $\mathcal{Z}$ and is given by $\mathcal{Z} = \{f(x) \in R^Q | x \in G(\mathcal{V},\mathcal{E})\}$.

Definition 2 *$f(x) \in \mathcal{Z}$ is a **nondominated** criterion vector iff x is an efficient solution to (4). Otherwise $f(x)$ is a **dominated** criterion vector.*

The set of all nondominated criterion vectors are denoted by $\mathcal{Z}_{Par}$ where $\mathcal{Z}_{Par} = f(\mathcal{X}_{Par})$. We use the Pareto optimality notation for both decision and criterion space.
Let S be a subset of $G(\mathcal{V},\mathcal{E})$. We will define the set of **locally efficient** solutions, denoted $\mathcal{X}_{Par}(S)$, to be the solutions that are efficient with respect to all other solutions in the subset S. Similarly, $\mathcal{Z}_{Par}(S)$ denotes the set of criterion vectors from $f(S)$ that are **locally nondominated** by any other criterion vector in $f(S)$.

2.1 Example

Now we present two small examples to illustrate the structure of the directed and the undirected problem, see Figure 2 and 3. Let the distance matrix $D_{directed}$ be given by

$$D_{directed} = \begin{bmatrix} 0 & 1 & 5 & 4 & 3 & 6 \\ 7 & 0 & 6 & 3 & 10 & 5 \\ 1 & 2 & 0 & 5 & 4 & 7 \\ 4 & 3 & 3 & 0 & 7 & 2 \\ 3 & 4 & 2 & 7 & 0 & 3 \\ 8 & 1 & 7 & 4 & 11 & 0 \end{bmatrix}$$

for the directed network of Figure 2. Let the weights be $w^1 = (-1,-2,-1,-1,-2,-2)$ and $w^2 = (2,1,2,2,2,1)$.
The solution procedure for the directed network in Figure 2 is explained in Section 3.2, and the criterion values are presented in Table 3.
Let the distance matrix D be given by

$$D = \begin{bmatrix} 0 & 1 & 1 & 4 & 3 & 2 \\ 1 & 0 & 2 & 3 & 4 & 1 \\ 1 & 2 & 0 & 3 & 2 & 3 \\ 4 & 3 & 3 & 0 & 5 & 2 \\ 3 & 4 & 2 & 5 & 0 & 3 \\ 2 & 1 & 3 & 2 & 3 & 0 \end{bmatrix}$$

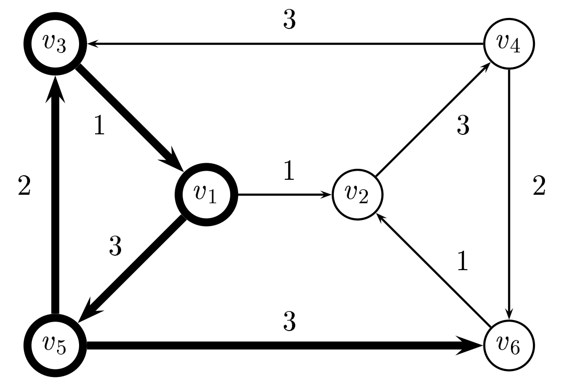

Figure 2: The directed network of Example 2.1. The bold parts constitute the set of efficient points.

for the undirected network of Figure 3. B can be calculated as

$$B = \begin{bmatrix} 2 & 3 & 3 & 6 & 1 & 4 \\ 3 & 2 & 4 & 1 & 6 & 3 \\ 2 & 3 & 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 & 4 & 3 \\ 2 & 3 & 1 & 4 & 1 & 4 \\ 3 & 2 & 4 & 1 & 4 & 1 \\ 4 & 3 & 3 & 4 & 1 & 2 \\ 3 & 2 & 4 & 3 & 2 & 1 \end{bmatrix}.$$

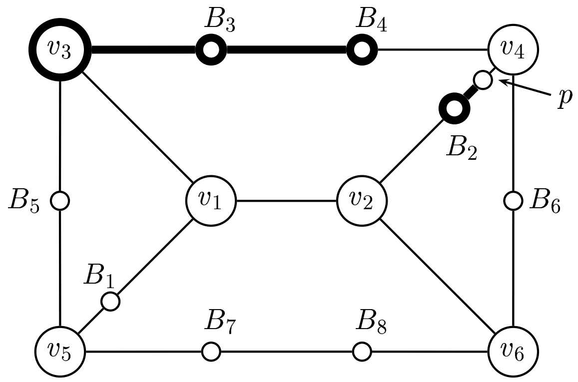

Figure 3: The undirected network of Example 2.1. The bold parts constitute the set of efficient points.

To clarify the solution to the undirected network in Figure 3 we present some function values in Table 2. The solution method for this bicriterion model is described in Section 4. Please note the values of p and B_4. This proves that a subedge, not having endpoint at a node or a bottleneck-point, can be efficient. We will refer to this example in Section 3 and 4.

Point x	$f(x) = (f^1(x), f^2(x))$
v_1	$(-17, 19)$
v_2	$(-16, 21)$
v_3	$(-18, 17)$
v_4	$(-27, 29)$
v_5	$(-24, 27)$
v_6	$(-15, 21)$
B_1	$(-27, 31)$
B_2	$(-30, 33)$
B_3	$(-25, 23)$
B_4	$(-28, 27)$
B_5	$(-23, 29)$
B_6	$(-20, 27)$
B_7	$(-25, 25)$
B_8	$(-23, 27)$
p	$(-28, 30\frac{1}{3})$

Table 2: Criterion values for all nodes, all bottleneck-points and point p.

From Table 2 we note that bottleneck-point B_2 is optimal for the maxisum criterion (f^1) and node v_3 is optimal for the minisum criterion (f^2).

3 General solution method for the Q criteria case

First, we solve two simple cases of the problem, namely the node problem and the directed case of the absolute location problem. Then we present the absolute location problem on an undirected network.

3.1 The easy case: $1/G, G_D/\bullet/d(\mathcal{V},\mathcal{V})/(Q_1\text{-}\sum_{obnox}, Q_2\text{-}\sum)_{Par}$

In this case the new facility can be placed only at the nodes of the given network, and we can determine the efficient set $\mathcal{X}_{Par} = \mathcal{X}_{Par}(\mathcal{V})$ by the following approach in $O(Qn^2)$ time, given the distance matrix D. This approach is presented in [5].

Algorithm 3.1:

1. $\mathcal{X}_{Par}(\mathcal{V}) = \mathcal{V}$;
2. for $i = 1$ to n do

   for $j = 1$ to n do

   if $f(v_j)$ dominates $f(v_i)$ then $\mathcal{X}_{Par}(\mathcal{V}) = \mathcal{X}_{Par}(\mathcal{V}) \setminus \{v_i\}$;

3. Output $\mathcal{X}_{Par}(\mathcal{V})$;

3.2 The easy case: $1/G_D/\bullet/d(\mathcal{V},G)/(Q_1\text{-}\sum_{obnox}, Q_2\text{-}\sum)_{Par}$

For this problem we have to investigate the objective function (2) of the directed case. First, we observe that the objective functions are constant on the interior of the edges. This is true because each term in the sum in (2) consists of a shortest cycle multiplied by a weight.

Theorem 1 *The directed objective function $f^q(x)$ defined in (2) is* ***constant*** *on $(e,(0,1))$ for all $e \in \mathcal{E}$ and for all $q \in \mathcal{Q}$.*

Proof :
Assume $e = (v_i, v_j) \in \mathcal{E}$. In the objective function

$$f^q(x) = \sum_{k=1}^{n} w_k^q \left(d(x, v_k) + d(v_k, x)\right) \quad q \in \mathcal{Q}$$

we observe that

$$\begin{aligned} d(x, v_k) &= d(x, v_j) + d(v_j, v_k) \quad \forall k \in \mathcal{V} \\ d(v_k, x) &= d(v_k, v_i) + d(v_i, x) \quad \forall k \in \mathcal{V} \end{aligned}$$

on the interior of e, and that

$$d(x, v_j) = (1-t)l(e) \quad \text{and} \quad d(v_i, x) = tl(e)$$

for some $t \in (0,1)$. After substituting the distance terms we get

$$f^q(x) = \sum_{k=1}^{n} w_k^q \left(d(v_j, v_k) + d(v_k, v_i) + l(e)\right) \tag{5}$$

which is independent of t, and thus of x, on the interior of e. ■

Next we use the triangular inequality to prove that the obnoxious objective functions, $q \in \mathcal{Q}_1$, have a higher value at the endnodes of e, and that the desirable objective functions, $q \in \mathcal{Q}_2$, have a lower value at the endnodes of e. To see this we analyze the objective function (2) once again.

Theorem 2 *Let $e = (v_i, v_j) \in \mathcal{E}$ be given. The obnoxious objective function values $f^q(v_i)$ and $f^q(v_j)$ are higher than $f^q(x)$, where x is an interior point on e for all $q \in \mathcal{Q}_1$.*

Proof :
WLOG we prove that $f^q(x) - f^q(v_i) < 0$. Remember that $w_i^q < 0, \forall i = 1, 2, \ldots, n$ and $q \in \mathcal{Q}_1$. Let us examine the two sums in

$$f^q(x) - f^q(v_i) = \sum_{k=1}^{n} w_k^q \,(d(x, v_k) - d(v_i, v_k)) + \sum_{k=1}^{n} w_k^q \,(d(v_k, x) - d(v_k, v_i)) \tag{6}$$

Starting at the second sum of (6) we use that $d(v_k, x) = d(v_k, v_i) + d(v_i, x)$ to get

$$\sum_{k=1}^{n} w_k^q \,(d(v_k, x) - d(v_k, v_i)) = \sum_{k=1}^{n} w_k^q \, d(v_i, x) = \sum_{k=1}^{n} w_k^q \, tl(e)$$

In the first sum of (6) we use the triangular inequality $d(v_i, v_k) \leq d(v_i, v_j) + d(v_j, v_k)$ and that $d(x, v_k) = d(x, v_j) + d(v_j, v_k)$. Remembering $w_i^q < 0$, we get

$$\begin{aligned}
\sum_{k=1}^{n} w_k^q \,(d(x, v_k) - d(v_i, v_k)) &= \sum_{\substack{k=1 \\ k \neq i}}^{n} w_k^q \,(d(x, v_k) - d(v_i, v_k)) + w_i^q \,(d(x, v_j) + d(v_j, v_i)) \\
&\leq \sum_{\substack{k=1 \\ k \neq i}}^{n} w_k^q \,(d(x, v_j) - d(v_i, v_j)) + w_i^q \,((1 - t)l(e) + d(v_j, v_i)) \\
&= \sum_{\substack{k=1 \\ k \neq i}}^{n} -w_k^q \, tl(e) + w_i^q \,((1 - t)l(e) + d(v_j, v_i)) \\
&= \sum_{k=1}^{n} -w_k^q \, tl(e) + w_i^q \,(l(e) + d(v_j, v_i)).
\end{aligned}$$

Hence,

$$f^q(x) - f^q(v_i) \leq w_i^q \,(l(e) + d(v_j, v_i)) < 0$$

because $w_i^q < 0$. The proof that $f^q(x) - f^q(v_j) < 0$ is similar, apart from the triangular inequality being used in the second sum of (6). ∎

Theorem 3 *Let $e = (v_i, v_j) \in \mathcal{E}$ be given. The desirable objective function values $f^q(v_i)$ and $f^q(v_j)$ are lower than $f^q(x)$, where x is an interior point on e for all $q \in \mathcal{Q}_2$.*

Proof :
Similar to the proof of Theorem 2, except $w_i^q > 0, \forall i = 1, 2, \ldots, n$ and $q \in \mathcal{Q}_2$. ∎

Using Theorem 3, we observe that the function values on $int(e)$ cannot dominate the function values at the nodes v_i and v_j, because the desirable function values at the nodes are lower. Similarly, the function values at the nodes cannot dominate the function value on the interior of e, because the obnoxious function value is lower on $int(e)$ by Theorem

2. This observation cannot, however, be used to conclude that nodes and edges cannot dominate each other. The objective function values on edge e_{12} in the directed network in Figure 2 are illustrated in Figure 4.

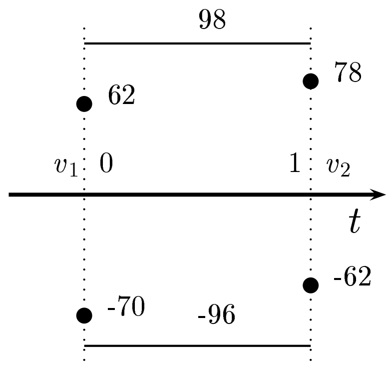

Figure 4: $f((v_1, v_2))$. Notice that $f(v_1)$ dominates $f(v_2)$.

In Algorithm 3.2 we have to compare all nodes and edges, but we only need one vector of function values on each edge, calculated easily by (5).
To present a compact form of the algorithm, we define the $n+m$ points a_i on $G(\mathcal{V}, \mathcal{E})$ as the n nodes and the midpoints on the m edges:

$$\begin{aligned} a_i &= v_i \quad \forall\, i = 1, 2, \ldots, n \\ a_{n+i} &= x_i = (e_i, \frac{1}{2}) \quad \forall\, i = 1, 2, \ldots, m \end{aligned}$$

Algorithm 3.2:

1. $\mathcal{X}_{Par} = G(\mathcal{V}, \mathcal{E})$;

2. for $i = 1$ to $n+m$ do

   for $j = 1$ to $n+m$ do

   if $f(a_j)$ dominates $f(a_i)$ then

   if $i \leq n$ then $\mathcal{X}_{Par} = \mathcal{X}_{Par} \setminus \{v_i\}$;

   if $i > n$ then $\mathcal{X}_{Par} = \mathcal{X}_{Par} \setminus (e_{i-n}, (0,1))$;

3. Output $\mathcal{X}_{Par}$;

When we make the pairwise comparison on the $n+m$ points, each taking $O(Q)$ time, we get a complexity bound of $O(Q(n+m)^2)$ time.

For the directed example in Figure 2, using (2) and (5), we get the criterion values of Table 3. The optimal value for the obnoxious function is -126 attained on (v_5, v_6) and the optimal desirable function value is 62 attained at v_1 and v_3. After running Algorithm 3.2 we have determined the efficient nodes and edges as indicated in the table and the figure.

Point x	$f(x) = (f^1(x), f^2(x))$	
v_1	$(-70, 62)$	Efficient
v_2	$(-62, 78)$	
v_3	$(-70, 62)$	Efficient
v_4	$(-68, 72)$	
v_5	$(-82, 80)$	Efficient
v_6	$(-74, 102)$	
(v_1, v_2)	$(-96, 98)$	
(v_1, v_5)	$(-94, 92)$	Efficient
(v_2, v_4)	$(-74, 84)$	
(v_3, v_1)	$(-76, 74)$	Efficient
(v_4, v_3)	$(-96, 98)$	
(v_4, v_6)	$(-98, 120)$	
(v_5, v_3)	$(-106, 98)$	Efficient
(v_5, v_6)	$(-126, 140)$	Efficient
(v_6, v_2)	$(-86, 108)$	

Table 3: Criterion values for all nodes and all edges.

3.3 Solving $1/G/\bullet/d(\mathcal{V}, G)/(Q_1\text{-}\sum_{obnox}, Q_2\text{-}\sum)_{Par}$

The general solution method consists of pairwise comparison of subedges. The objective functions are all piecewise linear, and the idea is to partition the network into subedges, where the objective functions are linear. The points where the piecewise linear functions change in slope are in fact the bottleneck-points. We then make a pairwise comparison of all these subedges and delete the inefficient parts. The result is the complete set of efficient solutions $\mathcal{X}_{Par}$.

It is important to note that part of a subedge may be efficient, starting at a point that is not a node or an edge-bottleneck-point (see Example 2.1 at point p).

For each comparison of two subedges we will construct a linear program to detect inefficient points (segments), that can be solved in linear time by methods found in Megiddo [9].

Let $z^q(t) = f^q(x_t)$, $x_t = (e, t)$. These Q functions are all piecewise linear with the same set of possible breakpoints corresponding to the bottleneck-points. Assume there are $P+1$

breakpoints including the two nodes. We then have P subedges. Let these breakpoints on (e,t) be denoted by $t_j,\ j=0,1,\dots,P,\ (1 \le P \le n-1)$, with $t_0 = v_i,\ t_P = v_j$ and $t_{j-1} < t_j\ \forall\, j = 1,2,\dots,P$. For $t \in [t_{j-1}, t_j]$, the $z^q(t)$'s are linear functions of the form

$$z^q(t) = m_j^q t + b_j^q \quad \forall\, q = 1,2,\dots,Q \quad \text{with}$$

$$\begin{array}{lll} m_1^q \le m_2^q \le \dots \le m_P^q\,, & b_1^q \ge b_2^q \ge \dots \ge b_P^q & q \in \mathcal{Q}_1 \\ m_1^q \ge m_2^q \ge \dots \ge m_P^q\,, & b_1^q \le b_2^q \le \dots \le b_P^q & q \in \mathcal{Q}_2 \end{array}$$

This is illustrated in Figure 1. Let us now compare the subedge A on edge e_A, $(e_A, [t_{j-1}, t_j])$ with subedge B on edge e_B, $(e_B, [s_{p-1}, s_p])$. A point $(e_A, t) \in (e_A, [t_{j-1}, t_j])$ is dominated by some point $(e_B, s) \in (e_B, [s_{p-1}, s_p])$ if and only if

$$m_p^q s + b_p^q \le m_j^q t + b_j^q \quad \forall\, q = 1,2,\dots,Q$$

where at least one inequality is strict. This comparison is illustrated in Figure 5 for two subedges from Example 2.1. Subedge (B_7, B_8) is compared with subedge (v_5, B_7).

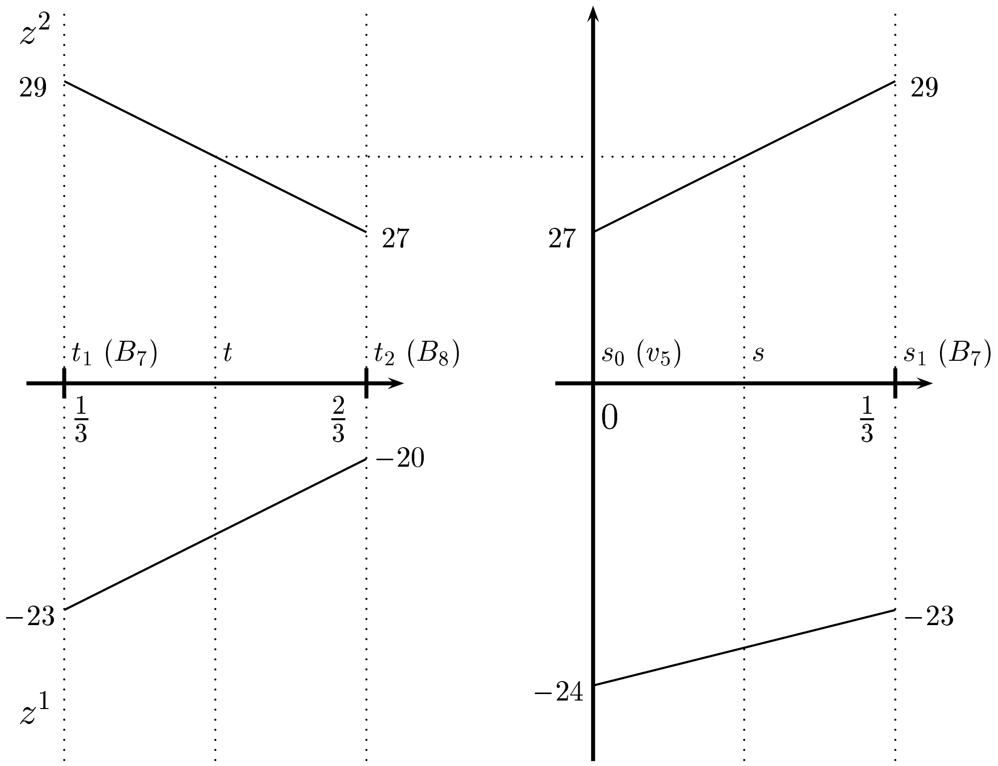

Figure 5: Comparing subedge (B_7, B_8) with subedge (v_5, B_7).

Let us define the set $\mathcal{T}$ where the inequalities hold (for these particular subedges) by

$$\mathcal{T} = \{(s,t)|\ m_j^q t - m_p^q s \geq b_p^q - b_j^q,\ \forall\, q \in \mathcal{Q}\} \cap ([s_{p-1}, s_p] \times [t_{j-1}, t_j])$$

If $\mathcal{T} = \emptyset$, $(e_B, [s_{p-1}, s_p])$ does not contain a point dominating any point in $(e_A, [t_{j-1}, t_j])$. Otherwise $\mathcal{T} \neq \emptyset$ is taken as a feasible solution set of the two 2-variable linear programs:

$$LB = \min\{\ t \mid (s,t) \in \mathcal{T}\} \quad \text{and} \quad UB = \max\{\ t \mid (s,t) \in \mathcal{T}\}$$

Using methods described by Megiddo [9], LB and UB can be calculated in $O(Q)$ time. We now check if we have only **weak dominance**. This means that none of the inequalities need to be strict as required by Definition 1. Note that points with weak dominated objective function values may be efficient. Let s_{LB} and s_{UB} be optimal values of s corresponding to LB and UB. These s-values are not necessarily unique as illustrated in Figure 6, where s_{LB} can be any point in $[0, \frac{1}{3}]$. In the case where s_{LB} (and/or s_{UB}) is not unique ($s_{LB} \in [s_a, s_b]$), we choose $s_{LB} = \frac{1}{2}(s_a + s_b)$ to avoid problems with weak dominance in the subedge endnodes. To check for weak dominance, we examine the subedge endnodes. If $m_p^q s_{LB} + b_p^q = m_j^q LB + b_j^q\ \forall\, q \in \mathcal{Q}$, then LB is only weakly dominated and can therefore still be efficient. Similarly, if $m_p^q s_{UB} + b_p^q = m_j^q UB + b_j^q\ \forall\, q \in \mathcal{Q}$, then UB is only weakly dominated. If both LB and UB are only weakly dominated, the entire subedge $(e_A, [t_{j-1}, t_j])$ is only weakly dominated by $(e_B, [s_{p-1}, s_p])$. This means that all the inequalities in $\mathcal{T}$ are in fact equalities. Otherwise the inefficient part of the subedge is deleted. If both LB and UB are dominated, then

$$(e_A, [t_{j-1}, t_j]) = (e_A, [t_{j-1}, t_j]) \setminus (e_A, [LB, UB])$$

and if, say LB is only weakly dominated, then

$$(e_A, [t_{j-1}, t_j]) = (e_A, [t_{j-1}, t_j]) \setminus (e_A, (LB, UB])$$

This comparison can also be done in linear time. The approach is simplified if one or both subedges consists of a single point (e_A, t') (or (e_B, s'')). If $(e_A, [t_{j-1}, t_j]) = (e_A, t') = x$, then $LB = UB = t'$ and

$$\mathcal{T}' = \{s|\ -m_p^q s \geq b_p^q - f^q(x),\ \forall\, q \in \mathcal{Q}\} \cap [s_{p-1}, s_p]$$

If $(e_B, [s_{p-1}, s_p]) = (e_B, s'') = y$, then

$$\mathcal{T}'' = \{t|\ m_j^q t \geq f^q(y) - b_j^q,\ \forall\, q \in \mathcal{Q}\} \cap [t_{j-1}, t_j]$$

and

$$LB = \min\{\ t \mid t \in \mathcal{T}''\} \quad \text{and} \quad UB = \max\{\ t \mid t \in \mathcal{T}''\}$$

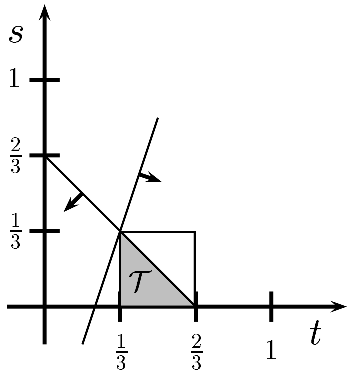

Figure 6: The linear programming constraints for comparing $(B_7, B_8) = (e, [\frac{1}{3}, \frac{2}{3}])$ with $(v_5, B_7) = (e, [0, \frac{1}{3}])$ on edge (v_5, v_6) in Example 2.1. $\mathcal{T}$ is indicated by the shaded area.

This subedge comparison is illustrated in Figure 6, where the subedge $(B_7, B_8) = (e, [\frac{1}{3}, \frac{2}{3}])$ from Example 2.1 is compared with $(v_5, B_7) = (e, [0, \frac{1}{3}])$. Both subedges are on the same edge. Since $\mathcal{T}$ is non-empty, we solve the two programs and find $LB = \frac{1}{3}$ and $UB = \frac{2}{3}$. Both LB and UB are dominated, so the subedge (B_7, B_8) is completely deleted.

Since we are removing a connected piece of $(e_A, [t_{j-1}, t_j])$, three things can happen. First, $(e_A, [t_{j-1}, t_j])$ can be completely deleted if $t_{j-1} = LB$ and $t_j = UB$ are both dominated. Second, a piece of $(e_A, [t_{j-1}, t_j])$ that includes one of the endpoints t_{j-1} or t_j can be deleted, in which case one connected subedge remains, say $(e_A, [t_{j-1}, LB))$ or $(e_A, [t_{j-1}, LB])$. The third case is when an interior part of $(e_A, [t_{j-1}, t_j])$ is deleted, so we end up with the two subedges $(e_A, [t_{j-1}, LB))$ and $(e_A, (UB, t_j])$, possibly including one of the points LB or UB. The third case is illustrated in Figure 7 where UB is not deleted, because $z(UB) = z(t_2)$.

In order to complete the comparison, we simply make an ordered subedge comparison. First, we compare $(e_1, [t_0, t_1])$ with all the other subedges, possibly dividing $(e_1, [t_0, t_1])$ into new subedges. Then we compare the second subedge $(e_1, [t_1, t_2])$ with all the remaining subedges. If $(e_1, [t_0, t_1])$ is not completely dominated, we also compare with this subedge. This comparison continues until we have compared the last subedge $(e_m, [s_{P-1}, s_P])$ with all the remaining subedges.

Notice that we can still use the entire subedge $(e_A, [t_{j-1}, t_j])$ to compare with the other subedges, even though a part of it is inefficient. It is only for the set of efficient points $\mathcal{X}_{Par}$, that we have to remember what part of $(e_A, [t_{j-1}, t_j])$ is efficient. But if the whole subedge $(e_A, [t_{j-1}, t_j])$ is inefficient, we should delete it from further consideration, also in the comparison process.

Assume that edge $e_i \in \mathcal{E}$ is divided into P_i bottleneck-point subedges.

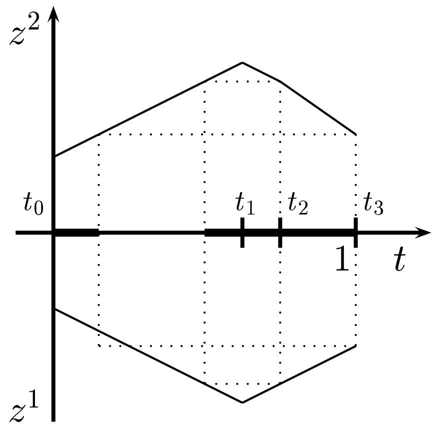

Figure 7: There are 4 breakpoints ($P = 3$) and 4 efficient subedges. Locally Pareto optimal subedges are indicated in bold on the t axes. Note that $(e, [t_2, t_3])$ dominates an interior part of $(e, [t_0, t_1])$.

Algorithm 3.3:

1. $\mathcal{X}_{Par} = G(\mathcal{V}, \mathcal{E})$;

2. for $i = 1$ to m do

   for $x = 1$ to P_i do

      for $j = 1$ to m do

      for $y = 1$ to P_j do

         compare $(e_i, [t_{x-1}, t_x])$ with $(e_j, [t_{y-1}, t_y])$

            $\mathcal{X}_{Par}$ unchanged if no points are dominated

            $\mathcal{X}_{Par} = \mathcal{X}_{Par} \setminus (e_i, [LB, UB])$ if LB and UB are dominated;

            $\mathcal{X}_{Par} = \mathcal{X}_{Par} \setminus (e_i, (LB, UB])$ if only UB is dominated;

            $\mathcal{X}_{Par} = \mathcal{X}_{Par} \setminus (e_i, [LB, UB))$ if only LB is dominated;

3. Output $\mathcal{X}_{Par}$;

This general algorithm has been implemented, and computational results are reported in Section 6. Each of the m edges may consist of up to $n - 1$ bottleneck-point subedges, giving at most $O(mn)$ subedges. If we make the global pairwise comparison on the $O(mn)$ bottleneck-point subedges, each taking $O(Q)$ time, we get a complexity bound of $O(Qm^2n^2)$ time. This is also the bound for the case where $\mathcal{Q} = \mathcal{Q}_2$ found in Hamacher *et al.* [5].

4 Bicriteria case

In the case where we only have two criteria, we may use the image of the network mapped into criterion space $\mathcal{Z}$ to solve the problem faster. This is done by calculating the lower envelope, see Hershberger [8]. This can be done in $O(p\ log\ p)$ time, where p is the number of line-segments. There are three different situations. $\mathcal{Q}_1 = \emptyset$ denoted min-min $(1/G/\bullet/d(\mathcal{V},G)/2\text{-}(\sum)_{Par})$, $|\mathcal{Q}_1| = |\mathcal{Q}_2| = 1$ denoted max-min $(1/G/\bullet/d(\mathcal{V},G)/(\sum_{obnox},\sum)_{Par})$ and $\mathcal{Q}_2 = \emptyset$ denoted max-max $(1/G/\bullet/d(\mathcal{V},G)/2\text{-}(\sum_{obnox})_{Par})$. All three cases are solved by the same method.

4.1 Direct mapping of the network into criterion space

This procedure is best described by an example, so we present the undirected network of Example 2.1 in criterion space.

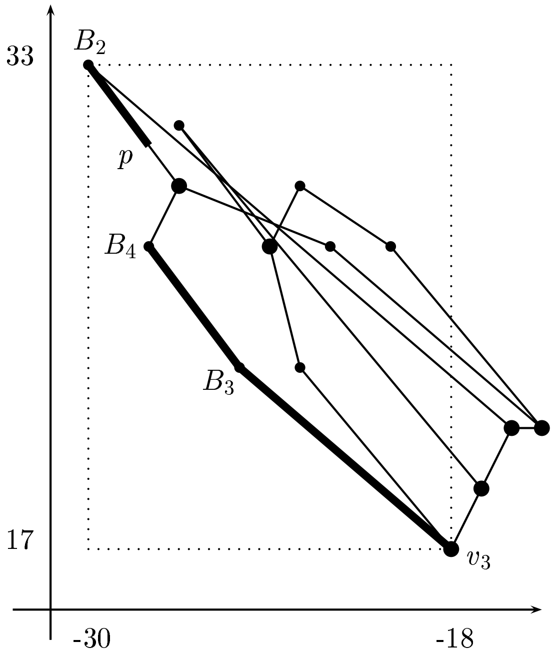

Figure 8: Mapping of the undirected network from Example 2.1 into criterion space. The bold parts constitute the set of nondominated points.

Since we want to find the set of efficient solutions $\mathcal{X}_{Par}$, we are only interested in values between the two extreme optimal solutions, namely $\mathcal{Z}^1$ and $\mathcal{Z}^2$. We therefore investigate the region $[f^1_{\mathcal{Z}^1}, f^1_{\mathcal{Z}^2}] \times [f^2_{\mathcal{Z}^2}, f^2_{\mathcal{Z}^1}]$, denoted S.

We have to make sure that the slope of the envelope is decreasing, when the f^1-values increase, to ensure that there are no dominated points on the envelope. This can be done by adding horizontal lines to all nodes and bottleneck-points in S, with the horizontal lines ending at $f^1_{\mathcal{Z}^2}$. This will at worst double the number of line-segments in the region S. Alternatively we could add the horizontal line to bottleneck-points that does not have a subedge with negative slope leaving the point. In the example of Figure 8 none of the points in S would need the horizontal line added. After the lower envelope is determined, we delete the horizontal parts (if any), because the points on a horizontal line are dominated by the left endpoint. The result is $\mathcal{Z}_{Par}$. The set of efficient solutions are then given by $\mathcal{X}_{Par} = f^{-1}(\mathcal{Z}_{Par})$. The efficient set corresponding to the nondominated set of Figure 8 is indicated in Figure 3.

We have the same complexity bound on the lower envelope calculation, as in Hamacher *et al.* [5], namely $O(mn\ log(mn))$. This bound can be rewritten by examining the *log* term and using the fact that m is at most n^2 for dense graphs. We therefore get the bound of $O(mn\ log\ n)$ time for the envelope calculation.

5 Center objectives - $1/G/\bullet/d(\mathcal{V},G)/(Q_3\text{-max}_{obnox}, Q_4\text{-max})_{Par}$

We now investigate the maximin and minimax objectives. These criterion functions are often referred to as the **weighted anti-center** and **center** of a network. The problem is formulated as follows:

$$\begin{array}{lll} \max & f^q(x) = \min_i\ w_i^q \cdot d(x,v_i) & q \in \mathcal{Q}_3 \\ \min & f^q(x) = \max_i\ w_i^q \cdot d(x,v_i) & q \in \mathcal{Q}_4 \\ \text{s.t.} & & \\ & x \in G(\mathcal{V},\mathcal{E}) & \end{array} \tag{7}$$

$\mathcal{Q}_3$ is the set of obnoxious objective functions, and $\mathcal{Q}_4$ is the set of attraction objective functions. At most one of the sets are allowed to be empty.

For simplicity we again multiply all objective functions in $\mathcal{Q}_3$ by -1 in order to minimize in stead of maximize. This gives the following formulation:

$$\begin{array}{lll} \min & f^q(x) = \max_i\ -w_i^q \cdot d(x,v_i) & q \in \mathcal{Q}_3 \\ \min & f^q(x) = \max_i\ w_i^q \cdot d(x,v_i) & q \in \mathcal{Q}_4 \\ \text{s.t.} & & \\ & x \in G(\mathcal{V},\mathcal{E}) & \end{array} \tag{8}$$

We notice that the objective functions are again piecewise linear, but the breakpoints are now weight dependent, see Figure 9. If we find these breakpoints, we can apply the same solution approach as in Section 3.3 for the multicriteria case, and the envelope method of Section 4 for the bicriteria case. When we only have center objective functions, the new breakpoints are the only ones needed. If we combine these objectives with the sum objectives, we may get a lot more breakpoints, because the bottleneck-point breakpoints are also needed.

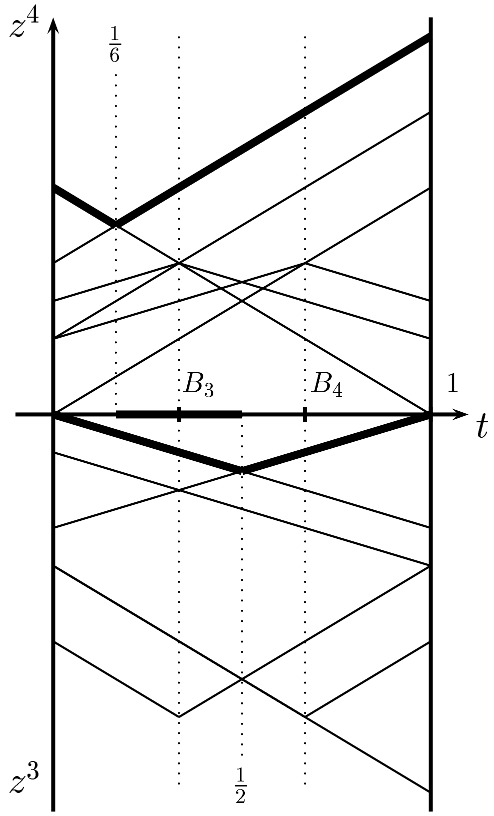

Figure 9: $f((v_3, v_4))$. There are two edge-bottleneck-points on this edge, and we find two new breakpoints. f^3 and f^4 are indicated with a bold lines.

In the following we expand Example 2.1 to illustrate what the center objectives look like. In Figure 9 we illustrate the locally efficient points on (v_3, v_4), where $w^3 = w^1$ and $w^4 = w^2$, as $\mathcal{X}_{Par}((v_3, v_4)) = ((v_3, v_4), [\frac{1}{6}, \frac{1}{2}])$.

In this example both objective functions turn out to be convex, but this is not the general case. The center objective is known to be neither convex nor concave. But the anti-center (maximin) objective is a **concave** function (so in problem (8) it is convex). This is true, because it is the minimum of piecewise linear concave functions. When we convert the problem to a minimax with negative weights, we get a piecewise linear **convex** function. This fact leaves little hope for finding an improved approach for this general case where we combine both sum and center objectives. After having investigated the different problems in turn, we can conclude that the method described in Section 3.3 works for any piecewise linear objective functions.

6 Computational results

In this section we present computational results from an implementation of Algorithm 3.3. We have not used the methods of Megiddo [9] in this implementation to solve the small LP's. Instead, we have used CPLEX 6.6. The code is programmed in C++ and the tests are run on a 700 MHz Linux PC.
We have used random networks of varying size generated using NETMAKER. A description of NETMAKER can be found in Skriver and Andersen [11]. All the random networks have a fixed number of nodes and a random number of edges with mean 4 times the number of nodes, i.e. a 50 node network has approximately 200 edges. Each network contains a random Hamiltonian cycle, and for each node three random edges are generated. The weights are generated negatively correlated. If one weight is in the integer interval from 1 to 33, the other is in the integer interval of 67 to 100. The same holds for the negative weights for the obnoxious objective functions (except for the sign). In each group we have used 10 random networks, and the mean is reported in the following tables.

First, we examine some semi-obnoxious bicriterion networks, having one push objective and one pull objective. The results are presented in Table 4. It appears that the number of subedges grows a little less than squared the number of nodes. The number of subedges is important, because in worst case we have to make a pairwise comparison of all these subedges, $(\#\text{ Subedges})^2$. The number of actual comparisons made is presented in the table, and the percentage of actual comparisons to the worst case is also presented. It is important to note that this percentage decreases as the networks increase in size.

# Nodes	50	100	150	200	250
CPU-time	40.96	229.54	774.64	1505.42	3326.37
# Subedges	3033.6	9411.5	18525.2	28368.1	39540.2
# Subedge comparisons (in millions)	0.358	1.770	5.138	8.655	16.531
# Efficient subedges	96.2	155.3	175.7	222.5	264.5
% Efficient subedges	3	1.6	0.95	0.78	0.67
% Comparisons	4.00	2.02	1.50	1.08	1.05
# Comparisons per sec	8733	7709	6633	5749	4970

Table 4: Semi-obnoxious bicriterion results, 1 push - 1 pull objective.

The number of efficient subedges is also presented in Table 4, and this number seems to grow linearly with the number of nodes. This number is in fact higher than the number of actual efficient subedges, because more subedges may contain the same efficient point, when this point is a node. If a node is efficient, all the subedges connected to this node contain some efficient points (perhaps only the node which is the endpoint of the subedge). The last row in Table 4 are the numbers of comparisons made per CPU-second. Assuming that CPLEX performs independently of the number of problems it has to solve, this decrease indicates that the large problems require a lot more storage of data, and accessing this data takes an increasing amount of time.

Next we examine the effect of having more objectives. These results are all computed on networks with 50 nodes. We reuse the results of the bicriterion (1-1) networks of Table 4, examine two types of three objective problems and one type of four objective problems. The three objective networks are generated with both 1 obnoxious and 2 desirable objectives (1-2), and 2 obnoxious and 1 desirable objectives (2-1). The four objective networks are all with 2 obnoxious and 2 desirable objective functions (2-2). The results are presented in Table 5.
As expected both the number of subedges containing efficient points and the CPU-time increase rapidly when more negatively correlated objective functions are added. With four objectives more than 75 % of the subedges contain efficient points. It is seen that the CPU-time for these instances is almost proportional to the number of subedge comparisons, since the data size of the instances is approximately the same (last line in Table 5).

# Objectives	1-1	1-2	2-1	2-2
CPU-time	40.96	123.05	105.49	870.57
# Subedges	3033.6	3293.1	3158.8	2853.6
# Subedge comparisons (in millions)	0.358	1.019	0.914	6.128
# Efficient subedges	96.2	359.1	357.9	2237.7
% Efficient subedges	3	11	11	78
% Comparisons	4.00	9.47	9.53	75.46
# Comparisons per sec	8733	8349	8720	7077

Table 5: The effect of having more objectives. All networks have 50 nodes.

Finally, we conclude that the computational results are constructive in the sence that rather large problems can be solved within a reasonable amount of time. Since location problems are usually not of the type you have to resolve often, a longer CPU-time is acceptable.
The most encouraging result being that for bicriterion networks with objective functions in almost opposite directions, a very small proportion of the networks is efficient. This indicates that this model is in fact an aid for the decision-maker, since a large part of the network can be omitted from further consideration. On the efficient parts of the network, the trade-off between the two objectives can then be revealed.

As a final comment, we note that with negatively correlated objectives, at most three objective functions should be considered. Otherwise the results are inconclusive, since a large proportion of the network will be efficient.

7 Concluding remarks

In this paper we have set up a multicriterion network location model for locating a (semi) obnoxious facility. We have proposed an efficient solution algorithm based on ideas from the multicriterion median network location problem presented in Hamacher *et al.* [5]. In the bicriterion case we have found an improved method, but this method has not been implemented. The general method presented in this paper works for all piecewise linear objective functions, and has been implemented in C++ using CPLEX as a solver. The computational results show that networks of realistic size can be solved in a reasonable amount of time. We thus conclude that this model is a good tool for general network location decisions.

References

[1] J. Brimberg and H. Juel. A bicriteria model for locating a semi-desirable facility in the plane. *European Journal of Operational Research*, 106:144–151, 1998.

[2] E. Carrizosa, E. Conde, and D. Romero-Morales. Location of a semiobnoxious facility. A biobjective approach. In 1996 Torremolinos, editor, *Advances in multiple objective and goal programming*, pages 338–346. Springer-Verlag, Berlin-Heidelberg, 1997.

[3] R.L. Church and R.S. Garfinkel. Locating an obnoxious facility on a network. *Transportation Science*, 12:107–118, 1978.

[4] M.S. Daskin. *Network and Discrete Location.* Wiley, New York, 1995.

[5] H.W. Hamacher, M. Labbe, and S. Nickel. Multicriteria network location problems with sum objectives. *Networks*, 33:79–92, 1999.

[6] H.W. Hamacher and S. Nickel. Multicriteria planar location problems. *European Journal of Operational Research*, 94:66–86, 1996.

[7] P. Hansen, M. Labbè, and J.F. Thisse. From the median to the generalized center. *RAIRO Rech. Opér.*, 25:73–86, 1991.

[8] J. Hershberger. Finding the upper envelope of n line segments in $O(n\ log\ n)$ time. *Info Process Lett*, 33:169–174, 1989.

[9] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput*, 12:759–776, 1983.

[10] A.J.V. Skriver and K.A. Andersen. A bicriterion semi-obnoxious facility location model solved by an ϵ-approximation. Technical Report 2000-1, Department of Operations Research, University of Aarhus, Denmark, 2000.

[11] A.J.V. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest path problems. *Computers and Operations Research*, 27:507–524, 2000.

[12] R.E. Steuer. *Multiple criteria optimization: Theory, Computation, and Application.* Wiley, New York, 1986.

[13] K. Thulasiraman and M.N.S. Swamy. *Graphs: Theory and Algorithms.* Wiley, New York, 1992.

Bicriteria Network Location (BNL) problems with criteria dependent lengths and minisum objectives

ANDERS J.V. SKRIVER AND KIM ALLAN ANDERSEN*
Department of Operations Research
University of Aarhus
Ny Munkegade, Building 530
DK - 8000 Århus C
Denmark

KAJ HOLMBERG
Department of Mathematics, Division of Optimization
Linköping Institute of Technology
S-581 83 Linköping
Sweden

August 17, 2001

Abstract

We present a new model, which is a generalization of the bicriterion median problem. We introduce two sum objectives and criteria dependent edge lengths. For this $\mathcal{NP}$ complete problem a solution method finding all the efficient solutions is presented. The method is a two-phases approach, which can easily be applied as an interactive method.

In Phase 1 the supported solutions are found, and in Phase 2 the unsupported solutions are found. This method can be thought of as a general approach to BOCO (Bi-objective Combinatorial Optimization) problems.

Keywords: MCDM, biobjective optimization, facility location, networks, MOCO.

1 Introduction

We begin by a motivating example. Assume we have to locate a money reserve, considering the two objectives of minimizing the transportation costs and the risk of having the transports robbed. The depot serves a number of clients varying in size, and we are given a connected network and interpret each of the n nodes as the clients. A relevant (node) weight for a client with respect to transportation costs is the number of monthly deliveries, and a weight for the risk objective is the maximum value of a money-transport. The

*Corresponding author. Email: kima@imf.au.dk

edge-lengths with respect to transportation costs could be the distance, and for the risk objective the edge-length could be the probability of an assault. If we assume that the cost of opening the new facility is independent of location, this particular cost is unimportant. A solution to this problem consists of two decisions. The first (and probably the most important) one is to decide where to locate the new facility (depot), and the second one consists in determining how to route the flow from the new facility to the nodes. The flow problem consists of $n-1$ Bicriterion Shortest Path (BSP) problems, which is a $\mathcal{NP}$ complete problem.
If each edge has only one length, we have the usual median problem. Now that we have one length for each criterion, the BSP problem becomes a subproblem. Therefore, this refinement has severe consequences on the complexity of the problem.

Before presenting the ideas behind the proposed solution method, some concepts from bicriterion analysis are reviewed. For a textbook introduction see Steuer [7] or Ehrgott [4]. Suppose we want to simultaneously minimize two functions $f^1(x)$ and $f^2(x)$ over some feasible set S. In our case S is a finite set of solutions.

$$\begin{array}{ll} \min & f^1(x) \\ \min & f^2(x) \\ \text{s.t.} & \\ & x \in S \end{array} \tag{1}$$

It is generally accepted, that solving (1) means finding the set of **efficient** (or Pareto optimal) solutions. A solution $x \in S$ is called efficient if one of the objective function values cannot be improved without worsening the other. Let $f(x) = (f^1(x), f^2(x))^t$, where t denotes transpose. The mathematical definition of efficiency is as follows.

Definition 1 *A point $x \in S$ is **efficient** iff there does not exist a point $\bar{x} \in S$ such that $f(\bar{x}) \leq f(x)$ with at least one strict inequality. Otherwise x is **inefficient**.*

Efficient points are defined in decision space. There is a natural counterpart in criterion space $Z = \{z \in \mathbb{R}^2 | \exists x \in S, z = f(x)\}$.

Definition 2 *$z(x) \in Z$ is a **nondominated** criterion vector iff x is an efficient solution. Otherwise $z(x)$ is a **dominated** criterion vector.*

In Definition 2 we have used that $z(x) = f(x)$. The set of efficient (E) solutions is denoted S^E, and the set of nondominated (ND) criterion vectors is denoted Z^{ND}, and is given by $Z^{ND} = z(S^E)$.

The criterion vectors can be partitioned into two kinds, namely supported and unsupported. Define the weighted objective function $W(x, \lambda)$ as:

$$W(x, \lambda) = \lambda f^1(x) + (1 - \lambda) f^2(x), \ \lambda \in (0; 1). \tag{2}$$

The function $W(x, \lambda)$ is a convex combination, or **weighted sum**, of the two objective functions. Optimizing this function over the feasible set S parametrically in $\lambda \in (0, 1)$ will give all the **supported** nondominated solutions to (1). The method is therefore often referred to as the **weighting method**.

It is important to note that each unsupported nondominated criterion vector is dominated by a convex combination of some set of nondominated criterion vectors. Supported nondominated (SND) criterion vectors are denoted Z^{SND} and the corresponding set of solutions are denoted S^{SE}.

The solution method proposed is a variant of the two-phases approach due to Ulungu and Teghem [9] and Visée *et al.* [10]. In Phase 1 all (or a representative subset of) the supported extreme solutions are found by using the weighting method. In Phase 2 a search between the supported solutions is conducted to find unsupported efficient solutions. The procedure is explained in details in Section 3.

The remaining parts of the paper is organized as follows. In Section 2 the bicriterion problem is presented, and some properties of the problem is given. In Section 3 the solution procedure is outlined, and an example is presented. In Section 4 the generalization to more than two criteria is discussed, and finally Section 5 contains the conclusions.

2 Problem formulation

We are given a connected directed network $G(\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ where $|\mathcal{V}| = n$ nodes, and edge set $\mathcal{E} = \{(v_i, v_j), (v_k, v_l), \dots, (v_p, v_q)\}$ with $|\mathcal{E}| = m$ edges. The underlying graph is denoted by G, and edges may be referred to by $e \in \mathcal{E}$, by $(v_i, v_j) \in \mathcal{E}$ or simply by $(i, j) \in \mathcal{E}$, where node i is the tail and node j is the head. Each node v_i carries two weights $(w_i^1, w_i^2)^t$, where $w_i^q \in \mathbb{R}_+$, $q = 1, 2$, so we may refer to the matrix of weights by $W_{2\times n}$. Each edge $e \in \mathcal{E}$ has length $l(e) = (l^1(e), l^2(e)) \in \mathbb{R}^2_+$. Let us define a matrix of edges $E_{m\times(4)}$ with the following entries. E_{i1} is the tail of edge e_i, E_{i2} is the head, $E_{i3} = l^1(e_i)$ is the length with respect to criteria one and $E_{i4} = l^2(e_i)$ is the length with respect to criteria two.

Notice that an undirected network can be modeled as a directed network with the double amount of edges. Define binary decision variables as follows:

$$x_i = \begin{cases} 1 & \text{if the facility is located in node } i \\ 0 & \text{else} \end{cases}$$

$$y_{ijk} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is used in the path to node } k \\ 0 & \text{else} \end{cases}$$

We examine the so-called median objectives or weighted sum objectives:

$$f^q(y) = \sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} l_{ij}^q w_k^q y_{ijk} \qquad q = 1,2$$

Combining the coefficients to $c_{ijk}^q = l_{ij}^q w_k^q$, we get

$$f^q(y) = \sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ijk}^q y_{ijk} \qquad q = 1,2 \tag{3}$$

There are two types of constraints. The first constraint ensures that exactly one facility is located and the second set of constraints ensures the existence of paths from the facility to the remaining nodes. This leads to the following problem:

$$\begin{aligned}
&\min \quad f^1(y) \\
&\min \quad f^2(y) \\
&\text{s.t.} \\
&\qquad \sum_{i=1}^{n} x_i = 1 \\
&\sum_{j=1}^{n} y_{jik} - \sum_{j=1}^{n} y_{ijk} = -x_i \qquad i \neq k, \quad \forall\, i,k \\
&\qquad x_i \in \{0,1\} \qquad \forall i \\
&\qquad y_{ijk} \in \{0,1\} \qquad \forall\, i,j,k
\end{aligned} \tag{4}$$

Notice that we have omitted the following redundant constraints

$$\sum_{j=1}^{n} y_{jik} - \sum_{j=1}^{n} y_{ijk} = 1 - x_i \quad \forall\, i, \text{ where } i = k.$$

The reason being that this part of the constraint matrix consists of n totally unimodular sub-matrices forming the n sets of paths, see (5). Notice that one path is non-existing, since the node in which the new facility is located, ships nothing through the network.

To understand the structure of the constraint matrix of (4), we write it out. We define the vector $\mathbf{y}_{ijk}$ (in bold) as the vector of all combinations of i and j, but with a fixed k. This way $\mathbf{y}_{ij1}$ contains all edge variables for node 1 and so forth. The matrix M_k is the totally unimodular sub-matrix forming paths from node x_i to node k. These matrices have dimension $(n-1) \times n^2$. I_{-k} is an $(n-1) \times n$ identity matrix with the k'th row deleted.

$$\begin{bmatrix} 1\cdots 1 & 0 & \cdots & 0 & \cdots & 0 \\ I_{-1} & M_1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ I_{-k} & 0 & \cdots & M_k & \cdots & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ I_{-n} & 0 & \cdots & 0 & \cdots & M_n \end{bmatrix} \begin{bmatrix} x \\ \mathbf{y}_{ij1} \\ \vdots \\ \mathbf{y}_{ijk} \\ \vdots \\ \mathbf{y}_{ijn} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5}$$

It turns out that this matrix is not totally unimodular.

Theorem 1 *The constraint matrix in (5) is not totally unimodular.*

An example of a sub-matrix of (5) with determinant two is given in the appendix. Since the constraint matrix is not totally unimodular, solving the LP relaxation of (4) is not guaranteed to return integer solutions, as is often the case in network problems.
Weighting the two objective functions in (4), using the weights λ and $1-\lambda$, $\lambda \in (0;1)$, results in the weighted version of (4)

$$\begin{array}{lrcll} \min & \lambda f^1(y) + (1-\lambda) f^2(y) & & & \\ \text{s.t.} & & & & \\ & \sum_{i=1}^{n} x_i & = & 1 & \\ & \sum_{j=1}^{n} y_{jik} - \sum_{j=1}^{n} y_{ijk} & = & -x_i & i \neq k \quad \forall\, i,k \\ & x_i \in \{0,1\} & & \forall i & \\ & y_{ijk} \in \{0,1\} & & \forall\, i,j,k & \end{array} \tag{6}$$

In Section 3.4 we describe how problem (6) can be solved in $O(n^4)$ running time using Benders' decomposition for a fixed λ.

3 Solution procedure

In this section the solution procedure for solving the bicriterion problem (4) is outlined. Before stating the procedure it may be helpful to consider a naive method. One possible way of solving the problem could be to solve problem (6) n times, namely one time for each possible location of the new facility. Suppose that the location of the new facility is fixed at a specific node, say node i (so $x_i = 1$). Using the weighting method, the supported efficient solutions (paths) with respect to node i can be revealed. We call these efficient solutions *locally* efficient with respect to node i. Given $\lambda \in (0,1)$ and x the corresponding locally efficient solution can be found in $O(n^3)$ running time, since we are faced with $n-1$ shortest path problems.

Finding the locally unsupported efficient solutions that are in fact globally efficient, constitutes a more difficult problem. These cannot be found using the weighting method. This fact is known from studying the BSP problem alone [5].

We thus have three types of efficient solutions:

- supported efficient solutions
- locally supported efficient solutions
- (locally) unsupported efficient solutions

The reason why locally supported efficient solutions are interesting, is that they may be unsupported efficient solutions in the main problem (4). These three kinds of solutions are illustrated in Example 3.1.

3.1 Example

We examine the network presented in Figure 1 with the following weights and edges. Each column of W consists of the two node-weights.

$$W = \begin{bmatrix} 200 & 300 & 500 & 100 & 400 & 500 & 400 \\ 7 & 4 & 2 & 6 & 6 & 2 & 8 \end{bmatrix}$$

The first two columns of E are the tail and head nodes. The next two columns are the two edge-lengths.

$$E = \begin{bmatrix} 1 & 2 & 78 & 22 \\ 1 & 3 & 24 & 72 \\ 1 & 4 & 26 & 71 \\ 1 & 5 & 13 & 71 \\ 1 & 7 & 86 & 12 \\ 2 & 3 & 98 & 29 \\ 2 & 5 & 17 & 90 \\ 3 & 5 & 29 & 97 \\ 3 & 6 & 87 & 28 \\ 3 & 7 & 7 & 69 \\ 4 & 5 & 4 & 77 \\ 4 & 7 & 89 & 5 \\ 5 & 6 & 17 & 92 \\ 5 & 7 & 40 & 74 \\ 6 & 7 & 69 & 12 \end{bmatrix}$$

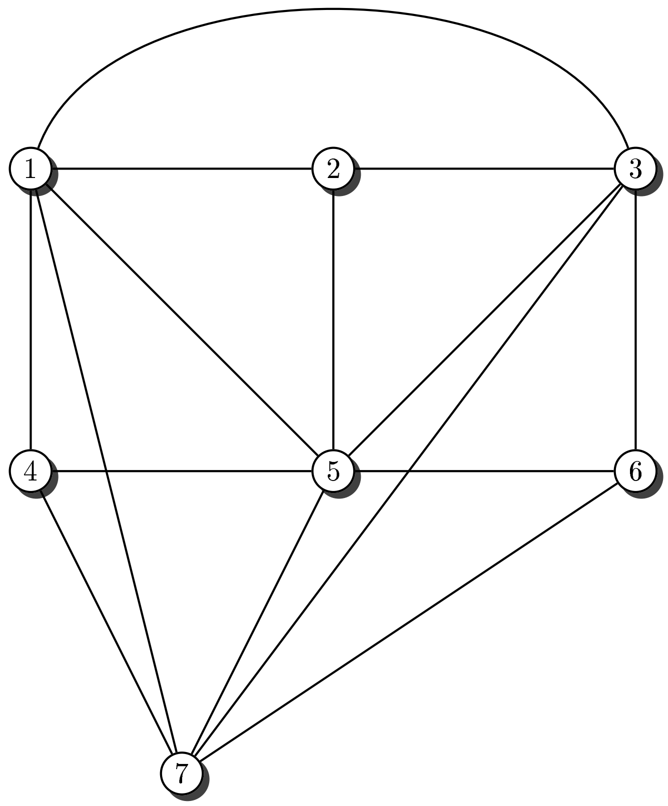

Figure 1: Network for Example 3.1.

The resulting 11 nondominated criterion vectors are presented in Table 1. These criterion vectors are visualized in Figure 2 and it is seen that there are 6 supported and 5 unsupported criterion vectors. Of the 5 unsupported solutions, only one, $(89200, 1868)$, is locally unsupported. The other four unsupported solutions are locally supported by the nodes indicated in Figure 2. The last nondominated solution, $(89200, 1868)$, is dominated by a convex combination of the following two locally supported solutions:

$$\frac{9}{11}(91200, 1684) + \frac{2}{11}(80200, 2587) = (89200, 1848.18)$$

There are a total of 2128 feasible criterion vectors, using only efficient paths between nodes. All these vectors are illustrated in Figure 3.

3.2 Two-phases approach

The procedure that we propose instead of the naive method, is a variant of the two-phases approach due to Ulungu and Teghem [9] and Visée *et al.* [10], and may be stated generically as:

- **Phase 1**: Find all (or a representative subset of) the supported solutions.

Node	f^1	f^2
5	45500	3025
5	47100	2289
1	78200	2062
7	89200	1868
7	91200	1684
1	92600	1506
7	97200	1376
1	107500	1182
7	111600	1112
7	129300	856
7	203800	798

Table 1: Nondominated values for Example 3.1.

- **Phase 2**: Conduct a search between the supported solutions in order to find unsupported nondominated solutions.

3.3 Phase 1

As explained in Section 2 all supported solutions to (4) may be obtained by solving the weighted program (6) parametrically in $\lambda \in (0,1)$. We will do that by employing NISE (Non-Inferior Set Estimation), a method presented in Cohon [3]. NISE guides the choice of $\lambda \in (0,1)$.

First, the weighted program (6) is solved using $\lambda = 1$ and $\lambda = 0$. This results in the minimum values f^{1*} and f^{2*} of the two objectives f^1 and f^2 respectively. Say there are alternative optima for the problem with $\lambda = 1$, then we choose a solution with the lowest objective function value of the second objective f^2. This automatically gives upper bounds, $\bar{f^2}$ and $\bar{f^1}$, on the other objective. The initial nondominated criterion vectors (in Z^{SND}) are $E_1 = (f^{1*}, \bar{f^2})$ and $E_2 = (\bar{f^1}, f^{2*})$.

Next we find the outward normal, $\bar{n} = (\bar{n}_1, \bar{n}_2)$, to the line between the two initial points, E_1 and E_2. Using $\lambda = \frac{\bar{n}_1}{\bar{n}_1 + \bar{n}_2}$ in solving (6), may result in two cases. We either get a new unique solution E_3, or we get E_1 or E_2 again. In the first case, the point E_3 is in Z^{SND}, and we continue by examining the two line-segments $E_1 - E_3$ and $E_2 - E_3$. In the latter case we know that there does not exist a supported (extreme) criterion vector between E_1 and E_2. The procedure proceeds until no new supported criterion vectors are found, or until a desired number of solutions are found. The outward normal to the line-segment between two points can easily be found as differences between the objective function values.

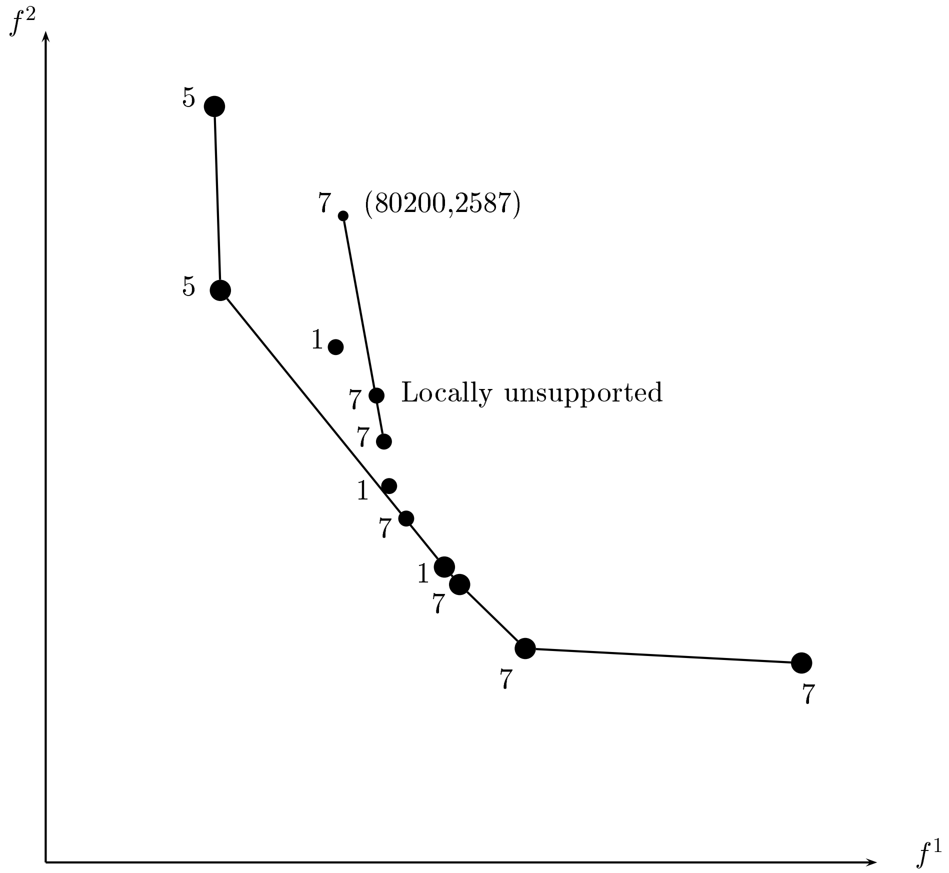

Figure 2: Nondominated vectors for Example 3.1. Large dots illustrate the supported solutions, and only one solution is locally unsupported. The numbers indicate the location node.

3.4 Benders' decomposition in Phase 1

In this section we present how Benders' decomposition can be used to find the supported solutions given a weight λ in Phase 1. Let λ be fixed and define

$$c_{ijk}(\lambda) = \lambda w_k^1 l_{ij}^1 + (1-\lambda) w_k^2 l_{ij}^2 \quad (\geq 0 \text{ since } l, w \geq 0).$$

When x is fixed, we can use the path constraints being totally unimodular, and relax the integrality constraints on y. Fixing x means locating the facility at a particular node. For a fixed $\overline{x}$ satisfying $\sum_i x_i = 1, \;\; x_i \in \{0,1\}$, we get the following **Benders' subproblem**:

$$\begin{aligned} \min \quad & \sum_{k,i,j} c_{ijk}(\lambda) y_{ijk} \\ s.t \quad & \\ & \sum_j y_{jik} - \sum_j y_{ijk} = -\overline{x}_i \quad i \neq k \quad \forall i,k \\ & 0 \leq y_{ijk} \leq 1 \qquad \forall i,j,k \end{aligned} \tag{7}$$

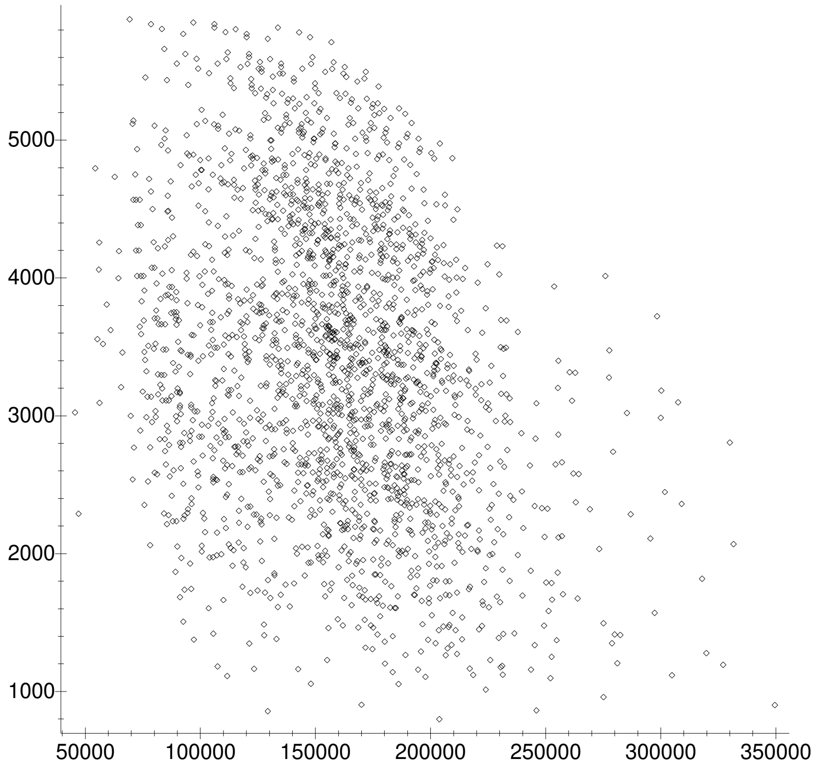

Figure 3: Illustration of 2128 criterion vectors for Example 3.1.

This linear programming problem has the following dual program:

$$
\begin{array}{lrcl}
\max & \sum_{\substack{i,k \\ i \neq k}} \alpha_{ik}(-\overline{x}_i) + \sum_{k,i,j} \beta_{ijk} & & \\
s.t & & & \\
 & \alpha_{jk} - \alpha_{ik} + \beta_{ijk} & \leq & c_{ijk}(\lambda) \quad i \neq k \quad \forall i,j,k \\
 & \beta & \leq & 0
\end{array}
\tag{8}
$$

The variables α are free variables corresponding to the path constraints in (7) and the β variables correspond to the upper bound on y. These dual variables can be found when the $n-1$ shortest path problems are solved in the Benders' subproblem, so we need not actually solve the dual problem (8). The dual leads to the following **Benders' master** problem:

$$\begin{array}{lll}
\min & v & \\
s.t & & \\
 & v \geq & -\sum\limits_{\substack{i,k \\ i \neq k}} \alpha_{ik}^l x_i + \sum\limits_{k,i,j} \beta_{ijk}^l \quad \forall l \\
 & \sum_i x_i = & 1 \\
 & x_i \in \{0,1\} & \forall i
\end{array} \tag{9}$$

where l is an index for the added inequalities.

The first time we generate a redundant inequality (or suggests a node picked earlier), the solution at hand is optimal (efficient). This is true because the subproblem (7) will return an earlier found solution.

Notice that Benders' master problem (9) is easy to solve in this case. It can be reformulated as a minimax problem. Let us rewrite the first constraint in (9), keeping in mind that only one x_i will be one.

$$\begin{aligned}
v &\geq -\sum_i \sum_{\substack{k \\ i \neq k}} \alpha_{ik}^l x_i + \sum_{k,h,j} \beta_{hjk}^l \\
v &\geq \sum_i \left(-\sum_{\substack{k \\ i \neq k}} \alpha_{ik}^l + \sum_{k,h,j} \beta_{hjk}^l \right) x_i \\
v &\geq \sum_i c_i^l x_i
\end{aligned}$$

where $c_i^l = -\sum\limits_{\substack{k \\ i \neq k}} \alpha_{ik}^l + \sum\limits_{k,h,j} \beta_{hjk}^l$. If we think of these c coefficients in a matrix, the optimal x_i is to find the column where the largest element c_i^l is as small as possible.

Notice, that we have to solve problems (7) and (9) at most $n-1$ times. Since Benders' subproblem consists of $n-1$ shortest path problems, problem (7) can be solved in $O(n^3)$ running time. Therefore the overall running time in Phase 1, given λ, is $O(n^4)$ running time.

3.5 Phase 2

Here we can first find the locally supported nondominated vectors by using the weighting method for a fixed node(s).

To find locally unsupported efficient points of (4), we use the Tchebycheff theory. Let $z = (z^1, z^2)$ denote a fixed reference point with $z \leq z^* = \left(f^{1*}, f^{2*}\right)$, where z^* is the ideal point. Then the augmented non-weighted Tchebycheff program (10) may be stated as

$$
\begin{array}{lrcll}
\min & \alpha+\rho\,\left(f^1(y)+f^2(y)\right) & & & \\
\text{s.t.} & & & & \\
& f^q(y)-\alpha & \leq & z^q & q=1,2 \\
& \sum_{i=1}^{n} x_i & = & 1 & \\
& \sum_{j=1}^{n} y_{jik}-\sum_{j=1}^{n} y_{ijk} & = & -x_i & i\neq k \quad \forall i,k \\
& x_i\in\{0,1\} & & \forall i & \\
& y_{ijk}\in\{0,1\} & & \forall i,j,k & \\
& \alpha\in\mathbb{R}_+ & & &
\end{array}
\tag{10}
$$

where ρ is a small positive constant ensuring that the solution found is in fact efficient. A few comments are in order. Note that instead of solving the usual weighted Tchebycheff program as found in Steuer and Choo [8], we propose to solve the augmented non-weighted Tchebycheff program (10). It was shown by Alves and Climaco [1] that all nondominated solutions to (4) can be found using the non-weighted program for integer problems (IP), and in Alves and Climaco [2] this result was generalized to mixed integer problems (MIP). Note that the augmented Tchebycheff program (10) has the same constraints as our original problem (4), as well as two additional constraints. The two new constraints are the reference point constraints, linking the reference point to the objective function in (10). These two new constraints complicate the problem, since they destroy the nice structure of the constraint matrix. Using Lagrange relaxation of these constraints does not solve our problem, as described in Appendix 2. We simply end up with the weighting method. However, problem (10) is a one objective MIP, which can be solved by the usual IP methods, such as branch and bound.

Next we explain how to determine the appropriate reference point(s). Assume that we want to search for locally unsupported solutions between the two nondominated points E_1 and E_2. First, we determine a maximum deviation factor

$$\delta=\max\left\{\delta^1,\delta^2\right\}$$

where $\delta^q=\bar{f}^q-f^{q*}$ $q=1,2$. This deviation factor is going to ensure that our reference point is below the ideal point z^*. Next we find reference points corresponding to our two nondominated solutions, E_1 and E_2:

$$z(E_i)=(E_i^1-\delta,E_i^2-\delta)\ i=1,2$$

The search reference point z_{new} can then be determined as the maximum of the reference point coordinates, because this point has a maximum distance of δ to both $z(E_1)$ and $z(E_2)$:

$$z_{new} = \left(\max\left\{z^1(E_1), z^1(E_2)\right\}, \max\left\{z^2(E_1), z^2(E_2)\right\}\right).$$

Using z_{new} in (10) can result in two things. If a new solution is returned, this solution is nondominated and defines two new search areas. Otherwise one of the points E_1 or E_2 is returned, and no nondominated (unsupported) solutions exist between the two points.

For our Example 3.1 we find $\delta = \max\{203800 - 45500, 3025 - 798\} = 158300$. Next we search for locally unsupported solutions between the two points $E_1 = (78200, 2062)$ and $E_2 = (91200, 1684)$ (on either side of the single locally unsupported point in Figure 2). This leads to the reference point $z_{new} = (-67100, -156238)$, where $\alpha = 158300$ can find both E_1 and E_2. In this case $E_3 = (89200, 1868)$ is found with $\alpha = 158106$.

4 Generalization to multiple criteria

Most of the ingredients in our approach easily generalize to more than two criteria. However, the NISE procedure used in Phase 1 to find supported nondominated points in a "spread-out" way, does not generalize. In two dimensions we find upper bounds on the objectives by minimizing the other objective alone. Forming the hyperplane between these two upper bounds, and then moving this hyperplane, we are guaranteed not to miss any supported nondominated solution. In three dimensions we may set upper bounds as the highest value from minimizing the other two objectives. The problem is that we may have supported nondominated solutions above this hyperplane. In Solanki *et al.* [6] these difficulties are explained.

Using another way to set the weights in Phase 1 in order to find the supported nondominated solutions, will leave us with a similar problem in Phase 2. Near the borders of the efficient frontier it may be difficult to determine a reference point in order to search for unsupported solutions.

5 Concluding remarks

In this paper we present a new, interesting location problem. This formulation incorporates both the location and the routing aspects in a multiobjective setting. We also present a solution method for the problem, and illustrate the problem structure and solution procedure by an example. The presented method can easily be made interactive, since the procedures in both phases are easily made interactive.

Appendix 1

Proof of Theorem 1:

Consider the complete directed network with 4 nodes ($n = 4$). This includes both directed edges between all nodes given k: (i, j, k) and (j, i, k) $\forall i, j, k$ where $i \neq j$. From (5) choose the first 4 columns corresponding to the x variables. Choose also the three columns corresponding to y_{124},y_{132} and y_{143}. Next we specify the seven rows. Choose the first row corresponding to the sum of x_i constraint. From I_{-4} choose rows 1 and 2, from I_{-2} choose rows 1 and 2 and from I_{-3} choose rows 1 and 3. This lead to the following 7×7 matrix with determinant two:

$$\begin{vmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{vmatrix} = 2$$

Appendix 2

Lagrange relaxation in the augmented Tchebycheff problem

As we will show, this approach does not help! We end up with the weighting method, if we relax the reference point constraints.

Let β be the Lagrange multiplier on the reference point constraints of problem (10). We are then left with the constraints of our original problem (4), and the constraint $\beta \geq 0$. Let's assume that β is fixed at $\bar{\beta}$. $\bar{\beta}$ can then be updated using for example a subgradient. The new objective function is given by

$$f(x, y) = \alpha + \rho \ \left(f^1(y) + f^2(y)\right) + \beta^1(f^1(y) - \alpha - z^1) + \beta^2(f^2(y) - \alpha - z^2).$$

Rearranging terms, we get

$$f(x, y) = (1 - \beta^1 - \beta^2)\alpha + (\rho + \beta^1)f^1(y) + (\rho + \beta^2)f^2(y) - \beta^1 z^1 - \beta^2 z^2 \qquad (11)$$

Let's evaluate the optimal value of α. If $1 - \beta^1 - \beta^2 \geq 0$, we choose $\alpha = 0$, and if $1 - \beta^1 - \beta^2 < 0$, we choose $\alpha = \infty$. Neither solution is good, because $\alpha = 0$ makes no improvement when we update $\bar{\beta}$ using the usual sub-gradient direction

$$d = (f^1(y) - \alpha - z^1, f^2(y) - \alpha - z^2)^t$$

Since z is a reference point $f(y) > z$, and we will simply increase $\bar{\beta}$ until we get the situation where $\alpha = \infty$. We therefore conclude that $\beta^1 + \beta^2 = 1$, so α can be any positive number. Since ρ is almost zero, we recognize this to be the weighting method applied in Phase 1.

References

[1] M.J. Alves and J. Climaco. Using cutting planes in an interactive reference point approach for multiobjective integer linear programming problems. *European Journal of Operational Research*, 117:565–577, 1999.

[2] M.J. Alves and J. Climaco. An interactive reference point approach for multiobjective mixed-integer programming using branch-and-bound. *European Journal of Operational Research*, 124:478–494, 2000.

[3] J.L. Cohon. *Multiobjective Programming and Planning.* Academic Press, 1978.

[4] M. Ehrgott. *Multicriteria Optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems.* Springer, 2000.

[5] A.J.V. Skriver and K.A. Andersen. A label correcting approach for solving bicriterion shortest path problems. *Computers and Operations Research*, 27:507–524, 2000.

[6] R.S. Solanki, P.A. Appino, and J.L. Cohon. Approximating the noninferior set in multiobjective linear programming problems. *European Journal of Operational Research*, 68:356–373, 1993.

[7] R.E. Steuer. *Multiple criteria optimization: Theory, Computation, and Application.* Wiley, New York, 1986.

[8] R.E. Steuer and E.U. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26:326–344, 1983.

[9] E.L. Ulungu and J. Teghem. The two-phases method: An efficient procedure to solve biobjective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20:149–165, 1995.

[10] M. Visée, J. Teghem, M. Pirlot, and E.L. Ulungu. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization*, 12:139–155, 1998.

Network planning in telecommunications: A stochastic programming approach

Morten Riis* and Anders J.V. Skriver
Department of Operations Research
University of Aarhus, Building 530
Ny Munkegade
DK - 8000 Århus C
Denmark

Jørn Lodahl
Sonofon
Skelagervej 1
DK - 9100 Ålborg
Denmark

August 16, 2001

Abstract

We consider a network design problem arising in mobile communications. The problem consists in deploying a number of new MSCs and allocating existing BSCs to MSCs, so as to minimize the incurred costs while meeting customer demand and observing the capacity restrictions. We formulate this problem as a two-stage stochastic program with mixed-integer recourse. To solve the problem we apply a dual decomposition procedure, solving scenario subproblems by means of branch and cut. The solution procedure has been tested on a real life problem instance provided by Sonofon, a Danish mobile communication network provider, and we report some results of our computational experiments.

Keywords: Network planning; Telecommunication; Stochastic Programming; Dual Decomposition; Branch and Cut.

1 Introduction

Mobile telecommunication network providers have been facing a rapid growth in demand for several years and this trend seems likely to continue. This forces the network provider

*Corresponding author. Email: riis@imf.au.dk

to constantly expand the capacity of the network in order to provide an acceptable grade of service to customers. There is a vast amount of literature concerning the optimal expansion of link capacities in a telecommunications network. We refer to papers by e.g. Balakrishnan, Magnanti and Wong [1], Bienstock and Günlük [2], Chang and Gavish [4] and Dahl and Stoer [5] for different approaches to such types of problems. The link capacities do not constitute the only potential bottleneck in a telecommunications network, however, since capacity restrictions may be imposed not only on traffic but also on the number of customers served by the network. In this paper we study a network design problem in which some capacity constraints are imposed to restrict traffic on links in the network while others are imposed to restrict the number of customers served by nodes in the network.

We study a mobile communications network. The base transceiver stations (BTSs) are each connected to one base station controller (BSC). Each BSC serves a number of BTSs and is connected to one mobile switching center (MSC). Finally each MSC serves a number of BSCs and the MSCs are connected internally. The network is illustrated in Figure 1.

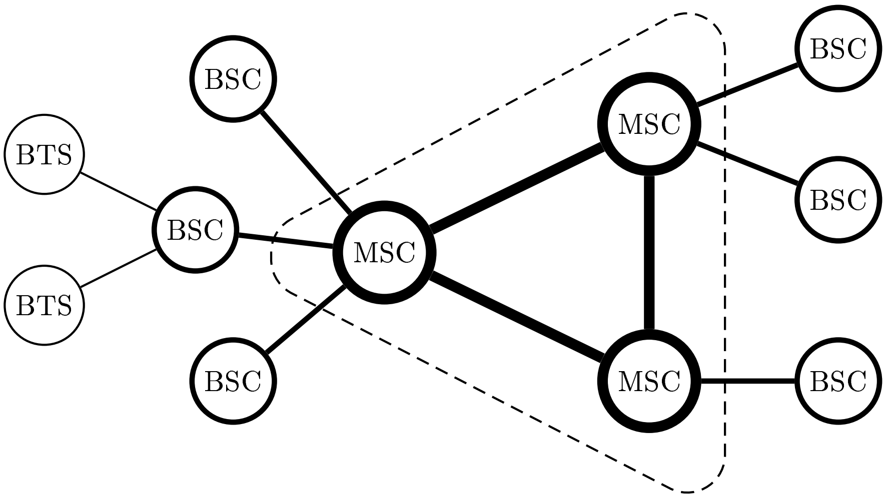

Figure 1: Illustration of a mobile telecommunications network.

The visitor location register (VLR) of an MSC, a database handling all information about clients, has a limited capacity, thus restricting the number of customers that can be served (through BTSs and BSCs) by an MSC. Thus the network provider not only has to expand the link capacities but should consider when and where to deploy new MSCs in order to be able to serve the increasing number of customers.

We will consider the problem of deploying a number of new MSCs and allocating the BSCs to new and existing MSCs, thus treating the number and locations of BTSs and BSCs as exogenous. The deployment of MSCs must be carried out so as to minimize the incurred costs while meeting customer demand and observing the capacity restrictions. The cost function will include four terms:

1. The cost of new MSCs.

2. The cost of connecting BSCs to MSCs.

3. The cost of expanding the capacity of links connecting the MSCs.

4. A penalty cost for handovers that occur among BSCs that are connected to different MSCs.

Tzifa et al. [17] study a similar problem in which only the access network is considered, thus ignoring the third cost term mentioned above. Also, the problem of optimally assigning BSCs to MSCs has been addressed by several authors such as Saha, Mukherjee and Bhattacharya [15] and Merchant and Sengupta [8]. Apart from minimizing the incurred costs of connecting BSCs to MSCs and the handover cost, it is customary to enforce some degree of load balancing among the MSCs. Tzifa et al. and Saha, Mukherjee and Bhattacharya explicitly include a penalty cost on uneven loads in the objective function, whereas Merchant and Sengupta propose to handle the load balancing problem parametrically. We do not explicitly consider load balancing but the parametric approach of Merchant and Sengupta may easily be adopted in our setting.
All of the above-mentioned authors follow a deterministic approach in the sense that the cost parameters, the number of customers and the demand for bandwidth are all assumed to be known at the point of decision. It is a fact, however, that the time that passes from the moment at which deployment of MSCs is resolved on, until the equipment is actually in place and available for use, is rather long (about a year). This means that at the time the decision has to be made, the network provider does not have full knowledge of several important parameters of the model. For this reason the network provider should put off the definitive decision on allocation of BSCs to MSCs for as long as possible, allowing uncertainty to be at least partially revealed. This is the incentive for us to model the problem as a two-stage stochastic program. In this formulation uncertain parameters are replaced by random variables and decisions are organized in two stages. The first stage consists of deployment of MSCs which must be resolved on before uncertainty has been revealed and hence must be based on the distribution of random parameters only. In the second stage outcomes of all random parameters have been observed and an optimal

allocation of BSCs to MSCs and a corresponding routing of traffic in the resulting network is determined.

The importance of including uncertainty in the problem formulation when modeling capacity expansion problems is well recognized. Stochastic programming has been used as a modeling tool for such problems in telecommunications by several authors. Sen, Doverspike and Cosares [16] study a capacity expansion problem in which the expected number of unserved requests is minimized subject to limitations on the total capacity expansion. Riis and Andersen [11, 12] use stochastic programming to solve two different capacity expansion problems in which additional capacity, required to meet customer demand, should be installed on edges of the network in modularities of fixed batch sizes. Finally, Dempster, Medova and Thompson [6] use chance-constrained programming to solve a capacity expansion problem subject to certain grade of service constraints assuming that the arrival process of calls is known. The main emphasis in previous studies has been on the capacity expansion of links, while less has been said about the network design problem considered in this paper.

This paper is organized as follows. We start out by formalizing the problem formulation and describing the parameters involved in Section 2. Extensions of the basic model to hedge against potential node and edge failures by imposing survivability constraints are discussed in Section 3. Next, in Section 4 we briefly outline the concept of dual decomposition (or scenario decomposition). Dual decomposition techniques have been applied in the context of stochastic programming by numerous authors including Carøe and Schultz [3], Mulvey and Ruszczynski [9] and Rockafellar and Wets [14]. The seminal idea is to use variable splitting to make the problem separable into independent subproblems which are easily solved. In our case, the subproblems are solved by means of branch and cut, using valid inequalities derived in Section 5 as cutting planes. In Section 6 our application is described along with some of the practical difficulties concerning implementation of the algorithm. Finally, we give some concluding remarks in Section 7.

2 Problem Formulation

To give a formal formulation of the capacity expansion problem introduced in the previous section, we will consider a finite number of potential locations for new MSCs and hence the basic setup will be described by three finite sets of nodes representing the locations of MSCs and BSCs:

- V_1 The set of locations of existing MSCs.

- $\cdot\, V_2$ The set of potential locations for new MSCs.
- $\cdot\, W$ The set of locations of BSCs.

Note that a given location may very well be represented as a node in more than one of the sets (even in all of them). In fact, the model allows for a single location to be represented as several nodes in one set, for example if we wish to deploy more than one MSC at a location.

The network interconnecting the MSCs is modeled as an undirected graph $G = (V, E)$. The nodeset $V = V_1 \cup V_2$ represents the existing and potential locations of MSCs, and the edge set E represents the existing and potential links $\{i, j\}$ between nodes $i, j \in V$. We will consider demand at BSC level. Even though we assume that traffic is bidirectional, we will find it convenient to use directed flow for modeling purposes. Hence we shall assign an arbitrary direction to each point-to-point demand and refer to its origin and destination. Also, each undirected edge $\{i, j\} \in E$ will correspond to two (conceptual) directed edges (i, j) and (j, i), each of which can carry flow. Still, to allow for the appropriate bidirectional traffic, edge capacities are dimensioned with respect to the total traffic on the given edge, disregarding the arbitrarily assigned directions of flow.

Demand for bandwidth on the connections will be described by a set K of commodities. Two main approaches for defining such commodities have been used in the literature. One possibility is to define a commodity for each point-to-point demand resulting in a total of $O(|W|^2)$ commodities. In general we find it more convenient, though, to reduce the number of variables by working with an aggregated formulation containing a total of only $O(|W|)$ commodities. This is achieved by letting each commodity $k \in K$ correspond to demand originating at a given BSC with respect to the arbitrary directions assigned to traffic. If one wishes to impose survivability constraints, however, it turns out that the disaggregated formulation may be more convenient. We will return to this issue in Section 3.

As previously discussed, several parameters of the model are not known with certainty at the time the decision on deployment of MSCs has to be made. In particular, the only information about future demand available at the point of decision, comes from past observations and some form of forecast model. This inherent uncertainty will be incorporated in the problem formulation by introducing some probability space $(\Omega, \mathcal{F}, P)$ and allowing the parameters in question to be dependent on the outcome of a random event $\omega \in \Omega$. Here, the probability distribution P is meant to reflect information about uncertain parameters coming from the above-mentioned forecasts. Thus the demand for bandwidth on edges and VLR-capacity at nodes will be described by the following sets of parameters:

- $D_{kr}(\omega)$ The net demand for commodity k at BSC r. $(k \in K, r \in W)$
- $L_r(\omega)$ The load of BSC r on the VLR in the MSC to which it is connected. $(r \in W)$

We emphasize that $D_{kr}(\omega)$ is the *net demand* for commodity k at BSC r and hence, in particular, that it is negative if and only if BSC r is the origin of commodity k and that $\sum_{r \in W} D_{kr}(\omega) = 0$. The parameter $D_{kr}(\omega)$ is directly related to the traffic between the origin of commodity k and BSC r, whereas the load $L_r(\omega)$ should rather be thought of as depending on the number of customers in the area served by BSC r.

Corresponding to the two types of demand, we have two types of existing capacity in the network - capacity restricting flow on edges of the network and capacity restricting the number of customers served by nodes in the network. These are summarized in the following sets of parameters:

- C_{ij} Flow-capacity on edge $\{i, j\}$. $(\{i, j\} \in E)$
- M_i VLR-capacity of the MSC located at node i. $(i \in V)$

The cost structure is described by the following sets of parameters some of which are treated as exogenous, while others are assumed to be uncertain at the point in time at which the decision has to be made, thus depending on the random event ω:

- c_i The cost of deploying an MSC at node i. $(i \in V_2)$
- $p_{ij}(\omega)$ The cost of adding one unit of capacity on edge $\{i, j\}$. $(\{i, j\} \in E)$
- $q_{ri}(\omega)$ The cost of connecting BSC r to node i. $(r \in W,\ i \in V)$
- $h_{rt}(\omega)$ The penalty cost (for supporting handovers) incurred if BSC r and t are connected to different MSCs. $(r, t \in W)$

Note that we assume the cost of expanding the capacity of a connection to be linear and that we do not include a fixed cost for establishing the connection. The reason for this is the fact that the company, in cooperation with which this research project was engaged upon, had already available a physical network with sufficient link capacities. In order to utilize this capacity, however, it may be necessary to install additional equipment at the end-points of the connection, and this cost is assumed to be linear with respect to the capacity provided.

The main decisions to be taken are deployment of new MSCs and allocation of BSCs to MSCs. These decisions are represented by the following two sets of binary variables:

$$\cdot\ x_i \quad = \begin{cases} 1 & \text{if an MSC is deployed in node } i.\ (i \in V_2) \\ 0 & \text{otherwise} \end{cases}$$

$$\cdot\ y_{ri}(\omega) = \begin{cases} 1 & \text{if BSC } r \text{ is connected to MSC } i.\ (r \in W,\ i \in V) \\ 0 & \text{otherwise} \end{cases}$$

As indicated by the dependency of the variables y_{ri} on the random event ω, the allocation of BSCs to MSCs is allowed to depend on the outcome of the random parameters. That is, the decision on allocation of BSCs to MSCs is postponed to the second stage to take full advantage of the additional information which is available at this point.

Finally, the following sets of variables are used to describe flow in the network, and the capacity expansion of links needed to carry this flow. Since flow does not occur until demand is realized, these variables all belong in the second stage.

- $f_{ijk}(\omega)$ Flow of commodity k on edge $\{i,j\}$ in direction from i to j. $(k \in K, \{i,j\} \in E)$

- $f_{jik}(\omega)$ Flow of commodity k on edge $\{i,j\}$ in direction from j to i. $(k \in K, \{i,j\} \in E)$

- $v_{ij}(\omega)$ Aggregate flow on edge $\{i,j\}$ in excess of current capacity C_{ij}. $(\{i,j\} \in E)$

To be capable of handling the model computationally, we will assume that there is only a finite number of possible outcomes of random parameters.

(A1) The probability distribution P is discrete and has finite support, say $\Omega = \{\omega^1, \ldots, \omega^S\}$ with corresponding probabilities $P\big(\{\omega^1\}\big) = \pi^1, \ldots, P\big(\{\omega^S\}\big) = \pi^S$.

A possible outcome of random parameters $(p(\omega^s), q(\omega^s), h(\omega^s), D(\omega^s), L(\omega^s))$ corresponding to some elementary event $\omega^s \in \Omega$ will be referred to as a scenario. For notational convenience we will refer to such a scenario simply by $(p^s, q^s, h^s, D^s, L^s)$. Likewise, we will use a superscript s on second-stage variables to indicate that these decisions are allowed to differ for different scenarios.

We are now ready to formulate the problem of optimally deploying a number of new MSCs and allocating BSCs to MSCs as a two-stage stochastic program. The first-stage objective is to minimize the sum of the cost of new MSCs and the expected value of the cost incurred in the second stage,

$$z = \min \sum_{i \in V_2} c_i x_i + \sum_{s=1}^{S} \pi^s Q^s(x) \tag{1}$$
$$\text{s.t. } x \in \mathbb{B}^{|V_2|}. \tag{2}$$

Here, the second-stage value function $Q^s(x)$ is given by

$$Q^s(x) = \min \sum_{\{i,j\} \in E} p_{ij}^s v_{ij}^s + \sum_{r \in W} \sum_{i \in V} q_{ri}^s y_{ri}^s + \sum_{\substack{r,t \in W \\ r<t}} h_{rt}^s \sum_{i \in V} (y_{ri}^s - y_{ti}^s)^+ \tag{3}$$
$$\text{s.t. } \sum_{r \in W} L_r^s y_{ri}^s \le M_i \qquad \forall i \in V_1, \tag{4}$$
$$\sum_{r \in W} L_r^s y_{ri}^s \le M_i x_i \qquad \forall i \in V_2, \tag{5}$$
$$\sum_{i \in V} y_{ri}^s = 1 \qquad \forall r \in W, \tag{6}$$
$$\sum_{j:\{i,j\} \in E} f_{jik}^s - \sum_{j:\{i,j\} \in E} f_{ijk}^s = \sum_{r \in W} D_{kr}^s y_{ri}^s \qquad \forall i \in V,\ k \in K, \tag{7}$$
$$\sum_{k \in K} \left(f_{ijk}^s + f_{jik}^s \right) \le C_{ij} + v_{ij}^s \qquad \forall \{i,j\} \in E, \tag{8}$$
$$y^s \in \mathbb{B}^{|W||V|},\ f^s \in \mathbb{R}_+^{2|E||K|},\ v^s \in \mathbb{R}_+^{|E|}. \tag{9}$$

We have used the notation x^+ to denote $\max\{0, x\}$ for $x \in \mathbb{R}$, and hence the third term of the second-stage objective (3) includes the handover cost between BSCs r and t if and only if these BSCs are allocated to different MSCs. The constraints (4) and (5) ensure that the total load from the BSCs connected to an MSC does not exceed the capacity of the VLR. Moreover, the constraint (5) ensures that a BSC can only be connected to an MSC if this is actually deployed (x_i=1) while the constraint (6) ensures that all BSCs are connected to exactly one MSC. The constraint (7) is a flow conservation constraint stating that the net flow of commodity k into MSC i should equal the aggregate net demand for commodity k from BSCs connected to MSC i. Finally, the constraint (8) states that the aggregate flow on an edge $\{i,j\} \in E$ cannot exceed the total capacity installed on the edge.

We note that the nonlinear term in the second-stage objective may easily be replaced by a linear one. Hence let H_{rt}^s be a variable representing the handover cost incurred between BSCs r and t under scenario s. Then H_{rt}^s may be defined using V linear constraints,

$$H_{rt}^s \ge h_{rt}^s (y_{ri}^s - y_{ti}^s) \qquad \forall i \in V, \tag{10}$$

and the nonlinear term may be replaced by a simple summation of the variables H^s_{rt}. Thus if the constraints (10) are added, the third objective term may be replaced by

$$\sum_{\substack{r,t \in W \\ r<t}} H^s_{rt}.$$

3 Survivability

There is an entirely different side to the issue of designing a telecommunications network under uncertainty besides the one we have considered this far. Thus it is possible that not only the parameters of the model, such as demand and prices, are subject to uncertainty. To be specific, we will consider a situation in which nodes and/or edges are subject to potential failures. This forces us to impose different kinds of *survivability constraints* to ensure that the network is not too vulnerable in case of such failures. The concept of survivability has previously been considered in the context of telecommunication networks by numerous authors. (See e.g. Dahl and Stoer [5] and Rios, Marianov and Gutierrez [13].) In general survivability may be achieved either by *diversification* or by *reservation* depending on the assurance required and the ability to restructure the solution in case of failures. In this section we discuss some possible formulations in the context of problem (1)-(9).

By diversification we mean routing demand using two or more edge- and/or node-disjoint paths. Diversification constraints are easily imposed if we are working with the disaggregate formulation in which each commodity $k \in K$ corresponds to a unique point-to-point demand. Hence we may let $O(k)$ and $D(k)$ denote the origin and destination of commodity k, and d^s_k the demand for commodity k under some scenario s so that D^s_{kr} equals d^s_k for $r = D(k)$, $-d^s_k$ for $r = O(k)$ and zero otherwise. If σ_k is a parameter equal to the maximum fraction of demand for commodity k that is allowed to flow through any given node or edge of the network, we may impose the following diversification constraints:

$$f^s_{ijk} + f^s_{jik} \le \sigma_k d^s_k \qquad \forall \{i,j\} \in E,\ k \in K \tag{11}$$

$$\sum_{j:\{i,j\}\in E} f^s_{ijk} \le \sigma_k d^s_k + (1-\sigma_k) d^s_k y^s_{O(k),i} \qquad \forall i \in V,\ k \in K \tag{12}$$

If paths are not required to be node disjoint the constraints defined by (12) are ignored. When working with the aggregate formulation on the other hand, we cannot impose such exact diversification constraints. One possibility is to use the following constraint, stating that at most a fraction of σ_k of the aggregate net flow of a commodity into a given MSC can arrive through one connection.

$$f^s_{jik} \le \sigma_k \sum_{r \in W} D^s_{kr} y^s_{ri} + \sum_{h:\{i,h\} \in E} f^s_{ihk} \qquad \forall \{i,j\} \in E,\ k \in K$$

As mentioned, another way to achieve survivability is by reservation. That is, to ensure the possibility of rerouting a given fraction of demand in the network resulting after a node or edge failure. To include reservation in the problem formulation each scenario should correspond not only to an outcome of the random parameters, but also to a specific failure state (possibly no failure). If all second-stage decisions may be modified in the light of a failure such an extension is easily included in the formulation, simply by modifying the node and/or edge set for each scenario according to the corresponding failure. It is more realistic, however, to assume that only rerouting of traffic is possible, whereas a swift reallocation of BSCs to MSCs or capacity expansion is not practicable. Such a situation would correspond to a three-stage stochastic program. In the first stage, as before, the deployment of MSCs is decided upon. In the second stage the outcome of random parameters is revealed and allocation of BSCs to MSCs and appropriate capacity expansion is carried out. Finally, in the third stage a failure possibly occurs and traffic is rerouted accordingly. Note that a node (MSC) failure in this situation would result in the loss of some demand, since BSCs allocated to the MSC in question would be cut off from the rest. We do not pursue this issue further in the present paper. It should be noted, however, that in theory such a three-stage problem could be solved by the solution procedure presented in the subsequent sections, but in practice the computational overhead involved would render such an approach intractable even for networks of moderate size.

4 Dual Decomposition

In this section we briefly outline the dual decomposition procedure which we are going to apply to problem (1)-(9). Dual decomposition, or scenario decomposition, exploits the fact that the vast majority of variables and constraints in the stochastic program are scenario dependent. In fact the only thing tying the scenarios together are the first-stage decisions on deployment of MSCs. Hence, if we use variable splitting on the first-stage variables, defining a deployment of MSCs for each scenario $x^1, \ldots, x^S$, problem (1)-(9) becomes separable into independent scenario subproblems. The fact that the deployment of MSCs cannot be scenario dependent may now be represented by a *non-anticipativity* *constraint* stating the problem as

$$\begin{aligned} z = \min & \sum_{s=1}^{S} \pi^s \Big(\sum_{i \in V_2} c_i x_i^s + Q^s(x^s) \Big) \\ \text{s.t. } & x^1 = \ldots = x^S, \\ & x^s \in \mathbb{B}^{|V_2|} \qquad \forall s \in \{1, \ldots, S\}. \end{aligned} \tag{13}$$

Relaxing the non-anticipativity constraint we obtain a problem which is completely separable into independent scenario subproblems. These subproblems are solved to obtain an optimal deployment of MSCs for each scenario. Next non-anticipativity is reinforced by branching on components of these solutions which differ among scenarios. To be specific, we introduce a branching tree initially consisting of only the root node corresponding to the original problem (13). In a given iteration we select a problem from the branching tree and solve the corresponding scenario subproblems obtaining scenario solutions $x^1, \ldots, x^S$. If MSC i is to be deployed in some scenario solutions and not in others, we add two problems to the branching tree imposing for $s = 1, \ldots, S$ the constraints $x_i^s = 0$ and $x_i^s = 1$ respectively. Otherwise, if all scenario solutions are equal, we have a feasible solution of the original problem and may update the upper bound if appropriate. For a thorough description of such a procedure, including a Lagrangian relaxation of the non-anticipativity constraints, we refer to Carøe and Schultz [3].

Clearly, if the scenario subproblems are solved by means of some branch and bound procedure, some effort should be taken to put information from previous iterations in the above procedure to use. Thus a node which is fathomed in a given subproblem in some iteration of the main procedure may be reconsidered in subsequent iterations since more variables are fixed as the main procedure progresses. In fact, for the problem instance considered in Section 6, the number of first-stage variables was so small (less than 20) that an enumeration tree could be created a priori and used for all scenarios, thus precluding any re-evalutions of nodes.

5 Valid Inequalities

In order to solve problem (1)-(9) using the dual decomposition procedure outlined in the previous section we need an efficient procedure for solving the scenario subproblems. To this end we will apply the concept of branch and cut which have proven to be a powerful tool for the solution of (mixed-) integer programming problems. As in ordinary branch and bound we start with the LP-relaxation of the mixed-integer programming problem and build a partitioning of the solution space in order to obtain an integral solution. The crucial idea in branch and cut is to combine this approach with a continuous generation of cutting planes tightening the formulation and thus reducing the size of the branching tree. For a thorough discussion of the branch and cut approach we refer to Padberg and Rinaldi [10] and Günlük [7]. As cutting planes we will use valid inequalities derived through simple polyhedral considerations.

First, we consider an inequality based on the total VLR-capacity installed through deployment of new MSCs. The inequality simply states that the total capacity of all VLRs in the resulting network should exceed the total demand from all BSCs. Formally the inequality is derived by summing the constraints (4)-(5), rearranging and rounding.

$$\sum_{i \in V_2} x_i^s \geq \left\lceil \frac{1}{M} \Big(\sum_{r \in W} L_r^s - \sum_{i \in V_1} M_i \Big) \right\rceil \qquad \forall s \in \{1, \ldots, S\}.$$

Here we have defined $M := \max_{i \in V_2} M_i$. Since the deployment of MSCs is not allowed to be scenario dependent this inequality may be strengthened further:

Proposition 1 *The following inequality is valid for the feasible region of all scenario subproblems, $s = 1, \ldots, S$.*

$$\sum_{i \in V_2} x_i^s \geq \max_{\tau \in \{1,\ldots,S\}} \left\lceil \frac{1}{M} \Big(\sum_{r \in W} L_r^\tau - \sum_{i \in V_1} M_i \Big) \right\rceil.$$

This inequality may be viewed as a global constraint in the sense that it is valid for all scenarios. As mentioned in the previous section we used an enumeration tree to solve subproblems for the instance considered in Section 6. Hence the above inequality was not actually included in the formulation but was merely used to reduce the size of the enumeration tree.

Next we consider a local constraint which is only guaranteed to be valid for the particular scenario from which it was derived. This inequality is based on the VLR-capacity of the individual MSCs and is used to enforce the fact that each BSC must be allocated to a unique MSC. Once again the underlying idea is simple. If the total demand from a group of BSCs exceeds the VLR-capacity of an MSC, we cannot allocate all of these BSCs to the MSC in question. This is formalized in the following proposition.

Proposition 2 *Let U be a subset of W such that $\sum_{r \in U} L_r^s > M_i$ for some MSC $i \in V$ and some scenario $s \in \{1, \ldots, S\}$. Then the following inequality is valid for the feasible region of the s'th scenario subproblem.*

$$\sum_{r \in U} y_{ri}^s \leq |U| - 1.$$

Naturally, this inequality will only be useful when the subset U of W is minimal in the sense that $\sum_{r \in U \setminus \{t\}} L_r^s \leq M_i$ for all $t \in U$, since it is otherwise dominated by other inequalities of the same type.

6 Numerical Results

In this section we will describe the practical application of our model. We have implemented our model on a real problem provided by Sonofon, a Danish mobile communication network provider. In this section we briefly describe the problem instance, the structure of costs and demand, and the practical collection and estimation of data. Due to competitive conditions, however, we cannot be too specific about the problem size and the input data. Finally, we report our computational results.

The problem under consideration has between 5 and 10 existing MSCs, less than 20 potential locations for new MSCs and less than 50 BSCs. The network interconnecting the MSCs is complete. The number of binary variables were reduced by dividing the area of interest into a number of regions and precluding from consideration certain allocations of BSCs to MSCs across regions. In the resulting formulation each scenario subproblem has 707 binary variables, 14598 continuous variables and 12045 constraints.

The cost of a new MSC is orders of magnitude higher than any other cost parameter. The cost of connecting a BSC to an MSC was set to zero if the BSC is currently connected to this particular MSC, and otherwise the total cost of a movement was estimated. Furthermore, the cost of expanding link capacities is given by the total cost of installing new equipment. The issue of determining an appropriate level for the artificial penalty cost for handovers, however, is a more complicated matter. Setting this level too low, may result in solutions with a large number of handovers which are not acceptable from a practical viewpoint. A high level, on the other hand, may result in configurations for which the gained practicability obtained by reducing the number of handovers is not sufficient to justify the increased cost. As a side effect computation time is likely to be increased in this case due to the large number of movements of BSCs required to reduce the number of handovers. In practice we chose to adjust the handover costs, observing their effect on solutions, so as to create geographically connected BSC areas.

The current demand for bandwidth and VLR-capacity was estimated from observations of traffic and the number of customers respectively. Future demand was then calculated using the estimates of current demand scaled by different scenario dependent growth factors. We have used the following procedure to generate demand for VLR-capacity at BSC r under scenario s,

$$L_r^s = \mu^s \cdot \rho_r^s \cdot L_r.$$

Here L_r is the current demand for VLR-capacity at BSC r, μ^s is a parameter reflecting the average growth in the number of customers, while ρ_r^s is a parameter reflecting regional fluctuations from this average growth. To capture the correlation between the demand for VLR-capacity and the demand for bandwidth, the net demand for commodity k at BSC r under scenario s was calculated using current demand D_{kr}, the above-mentioned parameters reflecting growth in the number of customers, and a third parameter σ^s reflecting growth in the demand for bandwidth per customer,

$$D_{kr}^s = \mu^s \cdot \sqrt{\rho_k^s \cdot \rho_r^s} \cdot \sigma^s \cdot D_{kr}.$$

Note that we have used the geometric average of the regional fluctuations ρ_k^s and ρ_r^s. Likewise the different cost terms were made scenario dependent by introducing stochastic fluctuations on future prices. The growth factors were all sampled from uniform distributions reflecting the expectations of Sonofon for the time horizon under consideration. As pointed out in Section 1, the second-stage decision of allocation of BSCs to MSCs is to be made after one year, and this was the time horizon used when estimating growth factors for the cost terms. As for customer demand, however, we have used a four-year time horizon when estimating the appropriate growth factors. This was done to ensure a somewhat stable solution guaranteeing sufficient network capacity for three additional years beyond the completed deployment of new MSCs. This means that demand is in fact only partially revealed at the time the second-stage decisions are to be made, but since the additional information obtained at this point will provide an improved estimate of the true rate of growth in demand, the gain of postponing some decisions to the second stage is likely to be considerable.

The algorithm was implemented in C++ using procedures from the callable library from CPLEX 6.6. Considering 100 scenarios, the solution times were about 3.5 hours CPU-time on a 700 MHz Linux PC. The solution suggested the deployment of one new MSC. Due to the complexity of the problem, the survivability constraints of Section 3 have not been implemented in the application. The valid inequalities of Section 5, however, have speeded up the solution times considerably.

7 Conclusions

In this paper we have set up a model for the optimal deployment of new MSCs in a mobile communications network. The model takes into account the cost of new MSCs, the cost of allocating BSCs to MSCs, and the cost of expanding capacities of links connecting the MSCs. Furthermore, a penalty cost was introduced to limit the number of handovers, inducing connected BSC areas. Since the deployment of MSCs involves a planning horizon of about a year, a number of important parameters of the model are not known with certainty at the point of decision. This lead us to a two-stage stochastic programming formulation of the problem. Considering 100 possible scenarios for the random parameters, the resulting formulation of a real-life problem contained more than a million variables and constraints and hence decompostion methods were called for. We chose to solve the problem using a dual decomposition procedure, solving scenario subproblems by means of branch and cut. The algorithm was implemented in C++ and the problem could be solved to optimality within a few hours of CPU time. We conclude that our model has been successfully implemented, and that it incorporates the most important details of the problem. We also conclude that the stochastic programming model is an important tool in the decision process, giving insight of the dynamics of the expansion problem.

References

[1] A. Balakrishnan, T.L. Magnanti, and R.T. Wong. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37:716–740, 1989.

[2] D. Bienstock and O. Günlük. Capacitated network design. Polyhedral structure and computation. *INFORMS Journal on Computing*, 8(3):243–259, 1996.

[3] C.C. Carøe and R. Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24:37–45, 1999.

[4] S.-G. Chang and B. Gavish. Telecommunications network topological design and capacity expansion: Formulations and algorithms. *Telecommunication Systems*, 1:99–131, 1993.

[5] G. Dahl and M. Stoer. A cutting plane algorithm for multicommodity survivable network design problems. *INFORMS Journal on Computing*, 10(1):1–11, 1998.

[6] M.A.H. Dempster, E.A. Medova, and R.T. Thompson. A stochastic programming approach to network planning. *Teletraffic Contributions for the Information Age. Proceedings of the 15th International Teletraffic Congress - ITC 15*, 1:329–339, 1997.

[7] O. Günlük. A branch-and-cut algorithm for capacitated network design. *Mathematical Programming*, 86:17–39, 1999.

[8] A. Merchant and B. Sengupta. Assignment of cells to switches in PCS networks. *IEEE/ACM Transactions on networking*, 3(5):521–526, 1995.

[9] J.M. Mulvey and A. Ruszczynski. A new scenario decomposition method for large-scale stochastic optimization. *Operations Research*, 43:477–490, 1995.

[10] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[11] M. Riis and K.A. Andersen. Capacitated network design with uncertain demand. Working Paper 2000/5, University of Aarhus, Department of Operations Research, 2000.

[12] M. Riis and K.A. Andersen. On using stochastic programming to plan the multiperiod capacity expansion of one connection in telecommunications. Working Paper 2000/2, University of Aarhus, Department of Operations Research, 2000.

[13] M. Rios, V. Marianov, and M. Gutierrez. Survivable capacitated network design problem: new formulation and Lagrangean relaxation. *Journal of the Operational Research Society*, 51:574–582, 2000.

[14] R.T. Rockafellar and R.J-B Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16:119–147, 1991.

[15] D. Saha, A. Mukherjee, and P.S. Bhattacharya. A simple heuristic for assignment of cells to switches in a PCS network. *Wireless Personal Communications*, 12:209–224, 2000.

[16] S. Sen, R.D. Doverspike, and S. Cosares. Network planning with random demand. *Telecommunication Systems*, 3:11–30, 1994.

[17] E.C. Tzifa, V.P. Demestichas, M.E. Theologou, and M.E. Anagnostou. Design of the access network segment of future mobile communications systems. *Wireless Personal Communications*, 11:247–268, 1999.

Solving Biobjective Combinatorial Max-Ordering Problems by Ranking Methods and a Two-Phases Approach

MATTHIAS EHRGOTT*
Department of Engineering Science
University of Auckland
Private Bag 92019
Auckland
New Zealand

ANDERS J.V. SKRIVER
Department of Operations Research
University of Aarhus, Building 530
Ny Munkegade
DK - 8000 Århus C
Denmark

August 16, 2001

Abstract

In this paper we propose a new method to solve biobjective combinatorial optimization problems of the max-ordering type. The method is based on the two-phases method and ranking algorithms to efficiently construct K best solutions for the underlying (single objective) combinatorial problem. We show that the method overcomes some of the difficulties of procedures proposed earlier. We illustrate this by an example and discuss the difficulties in extending it to more than two objectives.

Keywords: MCDM, biobjective optimization, max-ordering problems, ranking methods, combinatorial optimization.

1 Introduction

Max-ordering (MO) problems are multicriteria optimization problems in which the goal is to minimize the worst of several objective functions. They can be formulated as follows.

$$\min_{x \in S} \max_{i=1,\dots,Q} f_i(x), \tag{1}$$

where $f_i(x)$ denotes the objective functions of the problem. The problem is denoted max-ordering instead of min-max in order not to confuse terminology with single objective problems, i.e. $\min_{x \in S} \max_{e \in x} w_e$, which finds solutions where the largest weight is minimal, e.g. the path where the largest edge-weight is minimal. Max-ordering problems arise in various applications, see Rana and Vickson [23] or Warburton [29], and as subproblems in interactive methods for the solution of multicriteria optimization problems such as the

*Corresponding author. Email: m.ehrgott@auckland.ac.nz

GUESS method (Buchanan [3]), STEM (Benayoun et al. [2]), and the interactive weighted Tchebycheff method (Steuer and Choo [25]).

In this paper we consider max-ordering problems in a combinatorial context, i.e. we assume that S is a finite set, e.g. the set of paths between two nodes of a network, or the set of spanning trees of a graph.

There is a number of previous research papers on this topic (Ehrgott [5], Hamacher and Ruhe [15], Murthy and Her [21], Ehrgott et al. [9]). See also Ehrgott and Gandibleux [8] for more references. Various authors observed that, even in the bicriteria case, max-ordering problems are usually $\mathcal{NP}$-complete. The methods proposed for their solution include branch and bound (Rana and Vickson [23]), labeling algorithms (for shortest path problems, Murthy and Her [21]) and ranking methods (Ehrgott [5], Hamacher and Ruhe [15]), that is the application of algorithms to find K best solutions of (single objective) combinatorial problems.

We also propose methods involving ranking algorithms actually overcoming the main problem of the method proposed in Hamacher and Ruhe [15], at least for the case of two objectives, see the discussion after Algorithm 1. Our method also overcomes a weakness of the method proposed in Murthy and Her [21], see Section 4. We combine the ranking method with the two-phases method originally developed for the determination of all Pareto optimal solutions of bicriteria combinatorial optimization problems, Ulungu and Teghem [27], and so far, successfully applied to a number of such problems. We mention Ehrgott [6], Lee and Pulat [18] for network flow, Ulungu and Teghem [26] and Visée et al. [28] for knapsack, Ulungu and Teghem [27] for assignment, and Ramos et al. [22] for spanning tree problems.

2 Basic Results

In this section we introduce some notation for multicriteria (combinatorial) optimization and we prove some basic results which will justify the correctness of our method.

Consider a multicriteria optimization problem

$$\min_{x \in S}\{f_1(x), \ldots, f_Q(x)\}.$$

We use the notation $f(x) = (f_1(x), \ldots, f_Q(x))$ for the vector of objective functions. A feasible solution x^* is called Pareto optimal, if there is no $x \in S$ such that $f(x) \leq f(x^*)$ and $f(x) \neq f(x^*)$, where $\leq$ is understood component-wise. The set of Pareto optimal solutions of S is denoted $Par(S)$. If x^* is Pareto optimal, $f(x^*)$ is called efficient.

In multiobjective combinatorial optimization, Pareto optimal solutions can be classified into supported and unsupported Pareto optimal solutions. The former are those x^* for which there exists a weighting vector $\lambda = (\lambda_1, \ldots, \lambda_Q)$ such that

$$f(x^*) = \min_{x \in S} \sum_{i=1}^{Q} \lambda_i f_i(x).$$

The existence of unsupported Pareto optimal solutions is a characteristic property of multiobjective combinatorial optimization problems.

We shall also use the notation $g(x) = \max_{i=1,\ldots,Q} f_i(x)$ for the max-ordering objective value of a feasible solution $x \in S$. With these definitions we are ready to prove some basic results. The first one is wellknown, see e.g. Hamacher and Ruhe [15]. We state the proof for completeness.

Lemma 1 *There is at least one optimal solution of the max-ordering problem* $\min_{x \in S} g(x)$ *which is Pareto optimal.*

Proof : Suppose x^* is an optimal solution of the max-ordering problem, but is not Pareto optimal. Since S is finite, there must then exist a feasible solution $x \in S$ dominating x^*, i.e. such that $f_i(x) \leq f_i(x^*)$ for $i = 1, \ldots, Q$ with one strict inequality. Because $g(x) \leq g(x^*)$, it follows that x also solves the max-ordering problem optimally. ∎

The next Lemma is specifically stated for two objectives. It formalizes the argument that the maximum of two functions is minimal, if the objective values are as equal as possible. Its proof is immediate from the definition of the max-ordering problem and Lemma 1.

Lemma 2 *Let* $Par(S) = \{x_1, \ldots, x_p\}$ *be the set of Pareto optimal solutions of a bicriteria combinatorial optimization problem. Assume that* $f_1(x_i) \leq f_1(x_{i+1})$ *and* $f_2(x_i) \geq f_2(x_{i+1})$ *for* $1 = 1, \ldots, p-1$ *and define* $K := \min\{i : f_2(x_i) < f_1(x_i)\}$. *Then the following hold.*

1. *If* $K = 1$, x_1 *solves the max-ordering problem.*
2. *If* $K = \infty$, x_p *solves the max-ordering problem.*
3. *Otherwise* x_K *or* x_{K-1} *(or both) solve the max-ordering problem.*

A special case occurs if there is a Pareto optimal solution with both objectives equal.

Lemma 3 *If there is a Pareto optimal solution such that* $f_1(x) = f_2(x)$ *then* x *also minimizes* $g(x)$.

These three lemmas state that we can restrict our search for a solution for a minimizer of $g(x)$ to Pareto optimal solutions, with their two objectives as equal as possible. In other words, Pareto optimal max-ordering solutions will be located close to the halving line $f_1 = f_2$ in criterion space. Lemma 2 suggests to rank Pareto optimal solutions according to increasing values of f_1 (or f_2). This strategy would, however, imply the generation of supported and unsupported Pareto optimal solutions. And with the desired max-ordering solutions expected to be centrally located in the Pareto set, we would expect to enumerate half of all Pareto optimal solutions, involving excessive computational effort. Taking the difficulty of generating unsupported solutions into account (see Ehrgott [7]), we propose a different approach.
Our algorithm makes use of the information of Lemmas 1 to 3 in a more intelligent way and proceeds in two phases.

3 The Algorithm

First, we look for the two supported Pareto optimal solutions for which $f_1(x_i) \leq f_2(x_i)$ and $f_1(x_j) > f_2(x_j)$, $j > i$, according to the order of Lemma 2. We shall call them x_1 and x_2 in the algorithm. To do so, we start with solutions x_1 and x_2 minimizing objectives f_1 and f_2, respectively. We then proceed to solutions where the difference of objective values is smaller. When this is no longer possible, we will either have one supported Pareto optimal solution with $f_1(x) = f_2(x)$, or we end up with two neighboring supported Pareto optimal solutions, say x_1 and x_2 such that $f_1(x_1) < f_2(x_1)$ and $f_1(x_2) > f_2(x_2)$. According to Lemma 3, the first case solves $\min_{x \in S} g(x)$, and any other Pareto optimal solution must have one objective value smaller and one bigger than x. Of course, it may happen that one of the objectives dominates the other completely, i.e. $\min_{x \in S} f_1(x) \geq \max_{x \in Par(S)} f_2(x)$ (cases 1 or 2 in Lemma 2). In this case the problem is trivial, and we can easily detect it when computing x_1 and x_2 for the first time.
Should we terminate Phase 1 with two solutions, we will have to investigate unsupported solutions in the right-angled triangle defined by the hyperplane through the point $f(x^*)$ with normal λ and $(g(x^*), g(x^*))$, where x^* is the current best solution, see Figure 2. For this we use the ranking algorithm. In fact, $f(x_1)$ and $f(x_2)$ uniquely define weights λ_1, λ_2 such that both $x_1$and x_2 are optimal solutions of

$$\min_{x \in S} \lambda_1 f_1(x) + \lambda_2 f_2(x).$$

We can now apply a ranking algorithm to find second, third, ... best solutions for this problem, in order to find unsupported solutions in the identified triangle. A similar procedure was proposed for the identification of all unsupported Pareto optimal solutions in Coutinho-Rodrigues et al. [4].

The algorithm will stop if we encounter a solution x with $f_1(x) = f_2(x)$, as this must be the optimal solution we are looking for, or $\lambda_1 f_1(x) + \lambda_2 f_2(x) \geq g(x^*)$, because any further solutions will no longer be in the triangle and therefore no longer a candidate for a MO optimal solution. In the latter case, the currently best solution is the optimal solution of the max-ordering problem.

Algorithm - Phase 1

1. *Solve* $\min_{x \in S} f_1(x)$*, let* x_1 *be the optimal solution and let* $f_1^1 := f_1(x_1), f_2^1 = f_2(x_1)$*.*
2. *If* $f_1^1 \geq f_2^1$ *STOP,* $x^* = x_1$ *is an optimal solution.*
3. *Solve* $\min_{x \in S} f_2(x)$*, let* x_2 *be the optimal solution and let* $f_1^2 := f_1(x_2), f_2^2 = f_2(x_2)$*.*
4. *If* $f_2^2 \geq f_1^2$ *STOP,* $x^* = x_2$ *is an optimal solution.*
5. *If* $f(x_1) = f(x_2)$ *STOP,* $x^* = x_1$ *(or* x_2*) is an optimal solution.*
6. *Let* $x^* := argmin\{g(x_1), g(x_2)\}$ *be the currently best solution.*
7. *Let* $\lambda_1 := f_2^1 - f_2^2, \lambda_2 := f_1^2 - f_1^1$*.*
8. *Solve* $\min_{x \in S} \lambda_1 f_1(x) + \lambda_2 f_2(x)$*, let* x_3 *be the optimal solution and let* $f_1^3 := f_1(x_3)$*,* $f_2^3 = f_2(x_3)$*.*
9. *If* $f_1^3 = f_2^3$ *STOP,* $x^* = x_3$ *is an optimal solution.*
10. *If* $x_3 = x_2$ *or* $x_3 = x_1$ *call Phase 2(*λ_1, λ_2*).*
11. *If* $f_1^3 < f_2^3$ *then* $x_1 = x_3, f_1^1 = f_1^3, f_2^1 = f_2^3$*.*
12. *If* $f_1^3 > f_2^3$ *then* $x_2 = x_3, f_1^2 = f_1^3, f_2^2 = f_2^3$*.*
13. *Go to 6.*

The idea of the first phase is illustrated in Figure 1. With solutions x_1 and x_2 we compute the normal to the line connecting (f_1^1, f_2^1) and (f_1^2, f_2^2). This normal serves as a weighting vector for combining the two objectives, and its negative is the direction in which we search for a new supported Pareto optimal solution which is eventually found at x_3 with objective values (f_1^3, f_2^3).

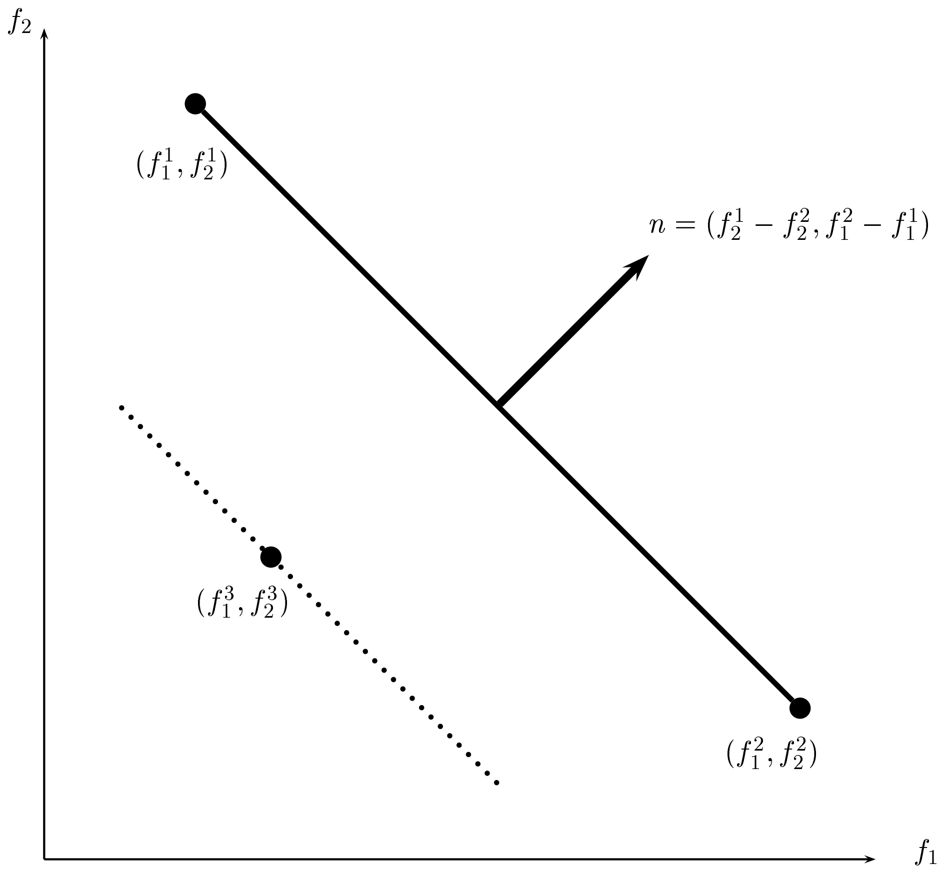

Figure 1: Illustration of Search Direction in Phase 1

We remark that the values λ_1, λ_2, identified at the end of Phase 1, are the best choice of λ in the method proposed by Hamacher and Ruhe [15] and will overcome the problem that for an unfortunate choice of λ, that method turns out to be complete enumeration of all feasible solutions.

Algorithm - Phase 2

1. *$K := 3$.*

2. *Use a K-best algorithm to find the K-best solution of $\min_{x\in S} \lambda_1 f_1(x) + \lambda_2 f_2(x)$. Denote this solution x^K.*

3. *If $\lambda_1 f_1(x^K) + \lambda_2 f_2(x^K) \geq g(x^*)$ STOP, x^* is an optimal solution.*

4. *If $f_1(x^K) = f_2(x^K)$ STOP, $x^* = x^K$ is an optimal solution.*

5. *If $f_1(x^K) > f_1^2$ then $K := K + 1$, go to 2.*

6. *If* $f_2(x^K) > f_2^1$ *then* $K := K + 1$, *go to 2.*

7. $K := K + 1$, *If* $g(x^K) < g(x^*)$ *then* $x^* := x^K$, *go to 2.*

We illustrate the algorithm on an example. In Figure 2 we show the objective values of 6 feasible points indexed in the order of their generation.

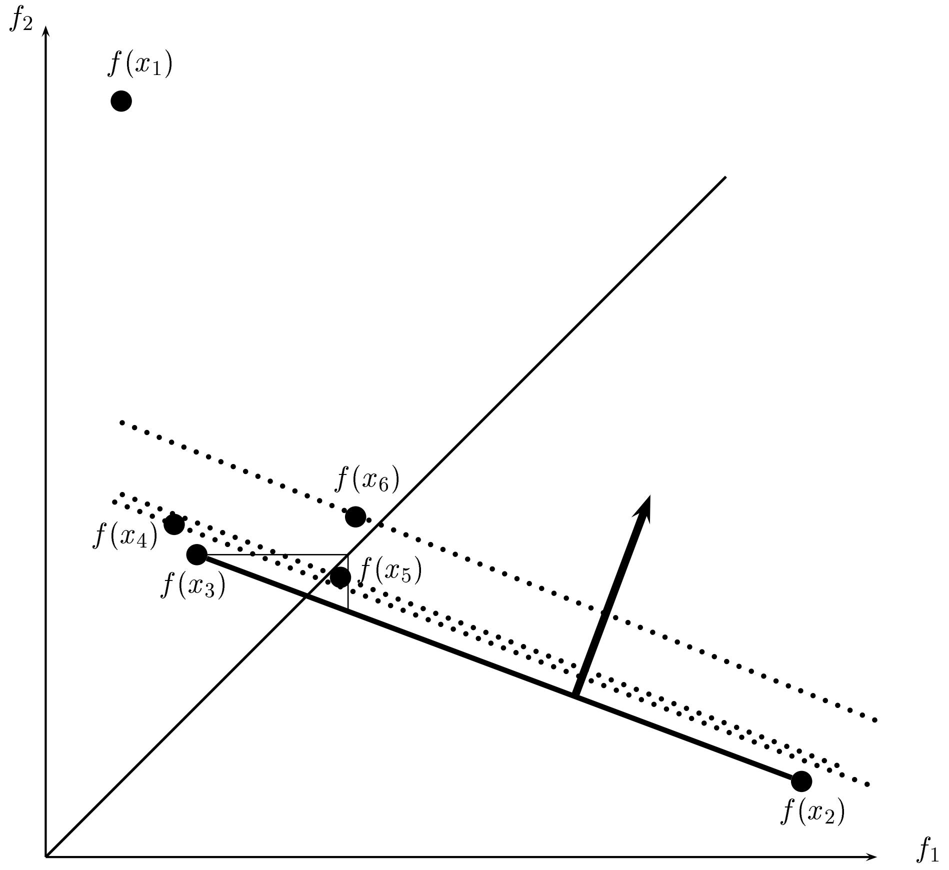

Figure 2: Illustrative Example

In Phase 1, x_1 and x_2 will be generated first. Weights λ_1 and λ_2 are computed corresponding to the normal to a line connecting $f(x_1)$ and $f(x_2)$ and $x^* = x_2$. Solution of the weighted sum problem in Step 8 results in x_3. Since $f_1(x_3) < f_2(x_3)$, f_1^1 and f_1^2 are replaced by the objective values of x_3. The current best x^* is updated to x_3. The second weighted sum problem uses updated λ's corresponding to the normal to the line connecting $f(x_2)$ and $f(x_3)$. Assume x_3 is returned as optimal solution. Thus no new supported Pareto optimal solution is found, and we continue with Phase 2 to investigate the earlier defined triangle. Note that the supported solution x_4 is not generated in Phase 1.

We know that x_3 and x_2 are first and second best solutions of the weighted sum problem, therefore we are searching for the third best by searching in direction λ. This turns out to be x_4, which is discarded as not being in the triangle $(f_2(x_4) > f_2(x_3) = g(x^*))$. So we set $K = 4$, identify x_5 as the next solution, which passes all tests. In our example x_5 replaces x_3 as the current best solution and K is set to 5. The next solution is x_6, the combined objective value of which is larger than that of the third corner point of the triangle. We will therefore find no further points in the triangle and stop with the optimal solution $x^* = x_5$.

Remark 1 *In Phase 2 the following situation might occur: The solution of the weighted sum problem is another supported Pareto optimal solution which is, as x_1 and x_2, optimal for the weighted sum problem. Its objective function vector lies on the line between $f(x_1)$ and $f(x_2)$. In this case, this point creates two new and smaller triangles. We can restrict search to the one which is intersected by the halving line $f_1 = f_2$.*

4 Lagrange Relaxation of Max-Ordering Problems

In this section we describe why Lagrange relaxation of max-ordering problems with linear objective functions does not work. This approach has earlier been suggested as a pruning method for a label correcting approach in Murthy and Her [21].
Consider the usual reformulation of (1)

$$\begin{array}{lrcl} \min & z & & \\ \text{s.t.} & z & \geq & f_i(x) \quad \forall\, i = 1, \ldots, Q \\ & x & \in & S \\ & z & \in & \mathbb{R}. \end{array} \tag{2}$$

A Lagrange relaxation of the first set of constraints in (2) is an appealing thing to do, as it simplifies the constraints to the original ones. This leads to the following problem, where λ is the vector of Lagrange multipliers:

$$\begin{array}{lrcl} \min & z & + & \sum_{i=1}^{Q} \lambda_i (f_i(x) - z) \\ \text{s.t.} & x & \in & S \\ & \lambda & \geq & 0. \end{array}$$

Rearranging the objective function leads to

$$\min_{x \in S} \left(1 - \sum_i \lambda_i\right) z + \sum_{i=1}^{Q} \lambda_i f_i(x),$$

where $\sum_i \lambda_i = 1$ to avoid an unbounded problem (since $z \in \mathbb{R}$). We thus end up with the following simple problem

$$\begin{array}{lrcl} \min & \sum_{i=1}^{Q} \lambda_i f_i(x) & & \\ \text{s.t.} & x & \in & S \\ & \lambda & \geq & 0 \\ & \sum_{i=1}^{Q} \lambda_i & = & 1. \end{array} \tag{3}$$

The multipliers are determined in the Lagrangian dual of (2), which has the objective function

$$\max_{\lambda} \min_{x \in S} \sum_{i=1}^{Q} \lambda_i f_i(x), \tag{4}$$

where the multipliers still have to fulfill the convexity constraints. (4) is easily solved by minimizing $f_i(x)$ for all i, and then setting $\lambda_i = 1$ for the largest $f_i(x)$.
We conclude that this approach will in fact return the worst possible Pareto optimal solution to our original problem (1) in the bicriteria case. With more than two objectives, worse solutions may exist.

5 K-best Algorithms

As we propose the use of ranking algorithms, our method is obviously restricted to such combinatorial optimization problems for which efficient methods for finding K-best solutions are available. We briefly review some of these here.
The largest amount of research on ranking solutions is available for the shortest path problem. Algorithms developed by Azevedo et al. [1], Martins et al. [19] or Eppstein [11] are very efficient. The best complexity known is $O(m+n\log n+K)$ by Eppstein's method. However, numerical experiments reported by Martins et al. [20] show their algorithm to be very competitive. Its complexity is $O(m + Kn\log n)$.
The second problem for which several methods are known, is the minimum spanning tree problem. We mention papers by Gabow [12] and Katoh et al. [16]. The best known complexity is $O(Km + \min(n^2, m\log\log n))$.
In the seventies and eighties some general schemes for ranking solutions of combinatorial optimization problems have been developed by Lawler [17] and Hamacher and Queyranne [14]. The application of the latter led to algorithms for matroids (Hamacher and Queyranne [14]), with the special case of uniform matroids discussed in Ehrgott [5]. The complexity of the latter is $O(K(n+m)+\min\{n\log n, nm\})$. Finally, an algorithm to rank (integer) network flows was presented in Hamacher [13]. Its complexity is $O(Knm^2)$.
We note that only algorithms allowing the construction of solutions with the same objective function values are applicable in our method. This is evident from the fact that at the

beginning of Phase 2, we have x_1 and x_2 as optimal, i.e. first and second best solutions of the weighted sums problem.

6 Discussion

The algorithm we propose solves the max-ordering problem for two criteria. It works efficiently, as it restricts search (in general) to a small subset of feasible solutions, where max-ordering solutions can be found. As it starts its search from supported Pareto optimal solutions which are much easier to generate than unsupported ones, it will in general enumerate only few solutions. It thereby resolves the difficulties of the ranking method proposed by Hamacher and Ruhe [15] in which the construction of an appropriate λ was an open question.

In addition, for large scale problems, when even the intelligent search applied in our algorithm might result in the enumeration of many feasible solutions (after all the problem is $\mathcal{NP}$-complete), the algorithm can be stopped at any time with the current best as an approximate solution. By computing $g(x^*) - g_{LB}$, where g_{LB} is a lower bound on g, we even have a bound on the distance from the real optimal solution. g_{LB} can easily be calculated and updated in Phase 1 in a straightforward manner. Initially, $g_{LB} = \max\{f_1^1, f_2^2\}$ with updates occurring whenever x_1 or x_2 is updated.

A natural question is the extension of the algorithm to more than two objectives. With such an endeavor we encounter two major difficulties. The first one is that problems with at least three objectives cannot be reduced to subproblems with two objectives only. Thus, in the multicriteria case all criteria have to be considered simultaneously.

Example 1 *Consider a combinatorial problem with three objectives and the following set of efficient vectors (objective vectors of Pareto optimal solutions)*

$$\left\{ \begin{pmatrix} 7 \\ 5 \\ 3 \end{pmatrix}, \begin{pmatrix} 6 \\ 4 \\ 8 \end{pmatrix}, \begin{pmatrix} 9 \\ 4 \\ 2 \end{pmatrix}, \begin{pmatrix} 6 \\ 8 \\ 2 \end{pmatrix} \right\}$$

The unique max-ordering solution is the first one, with $g(x) = 7$. However, looking at only two of the objectives at a time, we obtain the following. For f_1, f_2 only, the minimal value of $g(x)$ is attained at the second solution, for f_2, f_3 it is the third, and for f_1, f_3 it is the fourth. Thus none of the bicriteria subproblems yields the true optimal solution.

The second major difficulty is in the generalization of Phase 1. This problem has been observed by many researchers applying the method for the generation of all Pareto optimal solutions. In contrast to the bicriteria case, there may exist supported efficient points, which lie above (rather than below) a previously constructed hyperplane. For a discussion see Solanki et al. [24]. This kind of problem is very similar to the problem encountered in computing Nadir points for problems with at least three objectives see Ehrgott and Tenfelde [10] for a recent discussion. Further work is required to generalize our method in order to develop at least a heuristic to find a good λ in Phase 1 that will enable an efficient application of the ranking algorithms in Phase 2.

References

[1] J.A. Azevedo, M.E.O. Santos Costa, J.J.E.R. Silvestre Madeira, and E.Q.V. Martins. An algorithm for the ranking of shortest paths. *European Journal of Operational Research*, 69:97–106, 1993.

[2] R. Benayoun, J. de Montgolfier, J. Tergny, and O. Laritchev. Linear programming with multiple objective functions: Step method (stem). *Mathematical Programming*, 1(3):366–375, 1971.

[3] J.T. Buchanan. A naive approach for solving MCDM problems. *Journal of the Operational Research Society*, 48(2):202–206, 1997.

[4] J.M. Coutinho-Rodrigues, J.C.N. Climaco, and J.R. Current. An intercative bi-objective shortest path approach: Searching for unsupported nondominated solutions. *Computers and Operations Research*, 26(8):789–798, 1999.

[5] M. Ehrgott. On matroids with multiple objectives. *Optimization*, 38(1):73–84, 1996.

[6] M. Ehrgott. Integer solutions of multicriteria network flow problems. *Investigacao Operacional*, 19:229–243, 1999.

[7] M. Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transcations in Operational Research*, 7:5–31, 2000.

[8] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *Operations Research Spektrum*, 22:425–460, 2000.

[9] M. Ehrgott, S. Nickel, and H.W. Hamacher. Geometric methods to solve max-ordering location problems. *Discrete Applied Mathematics*, 93:3–20, 1999.

[10] M. Ehrgott and D. Tenfelde. Nadir values: Computation and use in compromise programming. Technical report, University of Kaiserslautern, Department of Mathematics, 2000. Report in Wirtschaftsmathematik Nr. 60/2000, submitted to European Journal of Operational Research.

[11] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[12] H.N. Gabow. Two algorithms for generating weighted spanning trees in order. *SIAM Journal of Computing*, 6(1):139–150, 1977.

[13] H.W. Hamacher. A note on K best network flows. *Annals of Operations Research*, 57:65–72, 1995. Special Volume "Industrial Systems".

[14] H.W. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4:123–143, 1985.

[15] H.W. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52:209–230, 1994.

[16] N. Katoh, T. Ibaraki, and H. Mine. An algorithm for finding k minimum spanning trees. *SIAM Journal of Computing*, 10(2):247–255, 1981.

[17] E.L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1972.

[18] H. Lee and P.S. Pulat. Bicriteria network flow problems: Integer case. *European Journal of Operational Research*, 66:148–157, 1993.

[19] E.Q.V. Martins, M.M.B. Pascoal, and J.L.E. Dos Santos. Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10(3):247–261, 1999.

[20] E.Q.V. Martins, M.M.B. Pascoal, and J.L.E. Dos Santos. A new improvement for a k shortest paths algorithm. Technical report, Universidade de Coimbra, 2000.

[21] I. Murthy and S.S. Her. Solving min-max shortest-path problems on a network. *Naval Research Logistics*, 39:669–683, 1992.

[22] R.M. Ramos, S. Alonso, J. Sicilia, and C. González. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111:617–628, 1998.

[23] K. Rana and R.G. Vickson. A model and solution algorithm for optimal routing of a time-chartered containership. *Transportation Science*, 22:83–96, 1988.

[24] R.S. Solanki, P.A. Appino, and J.L. Cohon. Approximating the noninferior set in multiobjective linear programming problems. *European Journal of Operational Research*, 68:356–373, 1993.

[25] R.E. Steuer and E.U. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26:326–344, 1983.

[26] E.L. Ulungu and J. Teghem. Application of the two phases method to solve the bi-objective knapsack problem. Technical report, Faculté Polytechnique de Mons, Belgium, 1994.

[27] E.L. Ulungu and J. Teghem. The two-phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1994.

[28] M. Visée, J. Teghem, M. Pirlot, and E.L. Ulungu. Two-phases method and branch and bound procedures to solve the bi-obective knapsack problem. *Journal of Global Optimization*, 12:139–155, 1998.

[29] A. Warburton. Aproximation of Pareto optima in multiple-objective shortest-path problems. *Operations Research*, 35(1):70 –79, 1987.

Summary

This thesis is on specific problems in the field of operations research, an area within mathematical economics. I focus on network and location problems combined with the area of multicriteria analysis. Multicriteria analysis is mathematical programming problems formulated with several (often conflicting) objective functions (goals). Therefore, the concept of optimality is broadened to Pareto optimality, also known from micro economics. With the concept of Pareto optimality a whole set of solutions can be "optimal", instead of just a single point. Since network and location problems are often combinatorial optimization problems, the complexity of the problems is an important issue. Many well-known combinatorial optimization problems are easy to solve (require polynomial solution time), but with more than one objective function they often become hard (require exponential solution time). In this thesis I develop methods for solving multicriteria network and location problems.

The thesis is built around seven papers, which should be read separately. The first part of the thesis is a seven chapter overview of my (and coauthors) work, and the second part contains the seven complete papers. Chapter 1 is a short introduction, Chapter 2 describes two closely related papers and the remaining five chapters cover one paper each.

Paper A is "A Classification of Bicriteria Shortest Path (BSP) Algorithms", which describes the four main solution approaches for the BSP problem. By examining the algorithmic structures, we argue why the Label Correcting method is believed to be the most effective method for this problem.
Paper B is "A label correcting approach for solving bicriterion shortest path problems", in which a preprocessing rule is introduced to the Label Correcting method to reduce the solution time. Computational experiments support the usefulness of the preprocessing rule. The last part of the paper contains a discussion on how to generate random networks for computational experiments on the BSP problem.
Paper C is "The Bicriterion Semi-obnoxious Location (BSL) Problem Solved by an ϵ-Approximation". This paper introduces a bicriterion model to describe the problem of locating a new airport. Two similar models are built for both the planar and the network case of the problem, and an approximation solution method is adapted. Computational experiments were performed on the real-life example of where to locate a new airport around the city of Aarhus, Denmark.
Paper D is "Multicriteria Semi-obnoxious Network Location (MSNL) Problems with Sum and Center Objectives". We present how a known, but rather new solution method works

by means of an illustrative example, and moreover discuss the generalization to a broader class of problems. The impact on the complexity of the algorithm by generalizing the problem is also presented. Finally a simple and very effective bicriterion approach is described and visualized.

Paper E is "Bicriteria Network Location (BNL) problems with criteria dependent lengths and minisum objectives" in which two well-known problems are combined. The result is a new problem, and possible applications are indicated. A two-phases solution method is adapted for the new problem, and the mathematical difficulties at different steps of the solution approach are discussed. The method is presented on an example illustrating the exact complications of the solution process.

Paper F is "Network planning in telecommunications: A stochastic programming approach" in which a capacity expansion problem arising in telecommunication is presented. The work is based on a research project with Sonofon. A stochastic programming model is built to describe when to expand capacity in order to meet required goals of service. Computational experiments were performed on data provided by Sonofon.

Paper G is "Solving Biobjective Combinatorial Max-Ordering Problems by Ranking Methods and a Two-Phases Approach", in which a two-phases solution method is adapted for the max-ordering problem. The max-ordering problem occurs as a subproblem in many well-known multicriteria solution methods. Therefore a good solution method for this problem is important. Ranking methods are used in Phase 2. The method is illustrated on an example.