Université Paris-Dauphine – École Doctorale de Dauphine
Laboratoire d'analyse et de modélisation de systèmes
pour l'aide à la décision

THESE

pour l'obtention du titre de :
DOCTEUR EN INFORMATIQUE
(arrêté du 7 août 2006)

Exact and approximate solving approaches in multi-objective combinatorial optimization, application to the minimum weight spanning tree problem

Approches de résolution exacte et approchée
en optimisation combinatoire multi-objectif,
application au problème de l'arbre couvrant de poids minimal

Candidat : Renaud Lacour

Directeur de thèse	Daniel Vanderpooten Professeur à l'Université Paris-Dauphine
Rapporteurs	Xavier Gandibleux Professeur à l'Université de Nantes
	Olivier Spanjaard Maître de conférences HDR à l'Université Pierre et Marie Curie
Examinateurs	Lucie Galand Maître de conférences à l'Université Paris-Dauphine
	Kathrin Klamroth Professeur à la Bergische Universität de Wuppertal (Allemagne)
	Jacques Teghem Professeur à la Faculté polytechnique de Mons (Belgique)

Présentée et soutenue publiquement le 2 juillet 2014

Remerciements

Je tiens tout d'abord à remercier Daniel Vanderpooten, mon directeur de thèse. Je lui dois en premier lieu le sujet de thèse qu'il m'a proposé. Son indéfectible soutien durant cette thèse m'a permis de la mener à bout. Il a su se rendre disponible pour de nombreuses réunions durant lesquelles nous avons eu de riches dicussions. Je le remercie de façon générale pour tout ce qu'il a fait pour moi, au-delà des aspects exclusivement liés à cette thèse.

Je remercie aussi Kathrin Klamroth, naturellement pour avoir accepté de faire partie de mon jury, mais aussi pour la collaboration que nous avons commencée durant son séjour au Lamsade au second semestre 2013. Celle-ci s'est manifestée par de nombreux échanges avec Daniel également, qui se sont vite révélés fructueux. Je remercie aussi Kathrin de m'avoir accueilli durant deux séjours à Wuppertal au cours desquels elle s'est montrée très disponible.

J'adresse ma reconnaissance à Xavier Gandibleux et Olivier Spanjaard qui ont accepté d'être rapporteurs et pour les échanges que nous avons eus, en particulier dans le cadre du projet GUEPARD. Je remercie également Olivier d'avoir fait partie de mon jury de pré-soutenance. Je remercie Lucie Galand et Jacques Teghem pour leur participation à mon jury en tant qu'examinateurs. La première soutenance de thèse à laquelle j'ai assisté était celle de Lucie, et je peux affirmer que sa thèse (Galand, 2008) est une référence pour de nombreux membres de la nouvelle génération à laquelle j'appartiens.

La présentation globale de ces travaux a été effectuée une première fois devant Peter Beisel et Kerstin Dächert, alors enseignants-chercheurs à l'Université de Wuppertal, et une seconde fois devant Mohamed Rahal, chercheur à ITE-VEDECOM. Je les remercie tous pour leurs retours qui ont contribué à la qualité de la présentation finale le 2 juillet 2014.

Ces années de thèse ont aussi connu de nombreux échanges avec plusieurs membres, permanents et doctorants, du Lamsade et du projet GUEPARD. Je pense à Yann Dujardin, avec qui j'ai eu de nombreuses discussions scientifiques, notamment mais pas uniquement en optimisation multi-objectif. J'ai beaucoup échangé aussi avec Lyes Belhoul, en particulier autour de nos sujets de thèse et notamment des aspects d'implantation que ceux-ci contiennent. Je remercie également Florian Jamain et Dalal Madakat pour nos discussions passionnées et pour leur enthousiasme perpétuel. Je salue enfin les nouveaux membres de l'« équipe multi-objectif » : Marek Cornu, Sami Kaddani et Satya Tamby (promis, après

la thèse je me mets sérieusement à Haskell !).

Je remercie l'équipe de l'école doctorale de Dauphine pour son travail au profit des doctorants et son aide tout au long de cette thèse. J'adresse également toute ma gratitude à l'équipe administrative et technique du Lamsade pour son soutien quotidien et sa sympathie. Je remercie également le service de la reprographie de Dauphine pour son ouverture au dialogue et son dévouement.

Je salue enfin mes collègues de bureau, présents et passés, en particulier Afef, Amine, Basile, Édouard, Liang-liang, Émeric et Sonia en plus de ceux que j'ai cités précédemment. La proximité géographique a permis de nombreuses discussions intéressantes.

Contents

Introduction **7**

1 Concepts and generic algorithms in MOCO **11**
1.1 Preliminary concepts 13
1.2 Implicit enumeration in MOCO 19
1.3 Exploration of the objective space 25

2 Some insights into the multi-objective Minimum Spanning Tree problem **31**
2.1 Notations 33
2.2 Computational hardness 34
2.3 From the single objective to the multi-objective case 35
2.4 Generic MOCO solving methods applied to the MOST problem 43
2.5 Ranking spanning trees 44

3 Determining the search region in multi-objective optimization **49**
3.1 Background and motivations 52
3.2 Existence and construction of upper bound sets 59
3.3 Properties of local upper bounds and their efficient computation 63
3.4 Complexity and computational experiments 70

4 Hybrid ranking and B&B algorithm for solving MOCO problems **79**
4.1 A ranking strategy 81
4.2 Integration of the ranking strategy in a branching tree 87
4.3 Computation of a reduced $(1+\varepsilon)$-approximation 91
4.4 Instantiation on the Minimum Spanning Tree problem and implementation issues 93
4.5 Computational experiments 97

Conclusions **111**

References **113**

List of Algorithms **123**

Nomenclature **125**

Introduction

Many real world decisions cannot be performed by comparing alternatives through a single value. Instead, it is more realistic to associate several values to each alternative, according to several criteria, points of view or states of the world. In the case of criteria, this gave rise to the subject of *multiple criteria decision making* (MCDM). MCDM mainly embraces *multiple criteria decision aid* (MCDA) and *multi-objective optimization* (MOO). While the former focuses on situations where the set of alternatives of a decision problem is defined in extension and is assumed to be of small cardinality, the latter faces a large, possibly infinite, set of feasible solutions. Therefore, most developments in MCDA focus on modeling issues, trying to determine and use in the best way the decision maker's preferences in order to formulate the most valuable recommendation regarding a practical decision problem. In contrast, MOO primarily faces computational issues arising from multiple and conflicting objective functions even when the single objective version of the problem is easily solved. MCDM in general finds numerous real world applications and so does MOO: Ehrgott (2008) details some of them in various fields such as finance, transportation, medicine, and telecommunication. Also Clímaco and Pascoal (2012) briefly survey recent applications in routing problems and telecommunication networks.

Several solution approaches exist in decision aid in general and therefore in MCDM in particular. The process can be more or less demanding for the decision maker: he can be involved only for a fixed number of phases or for an undetermined number of iterations in so-called interactive procedures. The solution to the problem with respect to the set of all possible alternatives can vary as well: it can be, according to Roy (1985), a selection of a subset of alternatives, a classification of the alternatives into categories, a ranking on all alternatives, or a description of all alternatives. We will consider only the first of these solution approaches in the remainder.

We now describe three possible types of choice subsets. A detailed survey on this subject can be found in Ehrgott and Wiecek (2005).

Determination of a single solution A first reaction when facing multiple objectives can be to aggregate them into one function, which maps each solution to a scalar value. The problem can then be solved as in single objective optimization. A wide variety of aggregation or scalarizing functions can be found in the literature, from weighted sum scalarization to

more complex non linear functions. Several aggregation functions aim at modelling complex decision maker's preferences such as the Choquet integral. Other, known as achievement functions, are used in interactive approaches, such as the weighted Tchebychev norm, whose optimization seeks the closest point to a reference point (see Miettinen and Mäkelä, 2002; Wierzbicki, 1986). The optimization of such non linear aggregation functions raises some difficulties but several efficient solution methods have been developed until recently, see e.g. Belhoul et al. (2014) and Galand et al. (2010). The use of an aggregation function may be combined with the requirement of a minimal outcome for each objective function. An example approach is to turn all but one objective into *budget*, resource, or knapsack constraints that are added to the problem while optimizing with respect to the remaining one.

Generation of all efficient solutions Another approach is the direct comparison of feasible solutions based on their outcomes. A central comparison criterion in MOO is Pareto dominance: a solution dominates another solution if it is at least as good on all obectives and better on at least one objective. According to this criterion, a decision maker would not be interested in a dominated solution, if we can show him instead a solution which dominates it. It is therefore interesting to generate all solutions that are not dominated, which are denoted *efficient* or *Pareto optimal*. The efficient set, i.e. the set of all efficient solutions, may be large in the case of highly conflicting objectives. Even a *reduced* efficient set, which contains all efficient solutions but keeps only one solution among equivalent solutions, can be large. It may in the worst case contain all feasible solutions. Still it provides in general an insight into the variety of outcomes that can be achieved by interesting feasible solutions; therefore the generation of the efficient set can be seen as a first step before the selection of a restricted set of solutions. This solution approach, which will be referred to as *exact* in the remainder, has been primarily developed for linear continuous problems (multi-objective linear programs, MOLP) and then also for combinatorial optimization problems (multi-objective combinatorial optimization, MOCO). While MOLP are easily solved, algorithms to solve MOCO have long been practically inefficient. Moreover, algorithms capable of solving MOCO problems have long been tied to the bi-objective case due to the more complex structure of problems with more than two objectives. The proposed algorithms in the literature however improved a lot during the last decade. For example, Sourd and Spanjaard (2008) showed that multi-objective branch and bound can efficiently solve large instances of the bi-objective Minimum Spanning Tree problem. Also Przybylski et al. (2010b) proposed a multi-objective method, not limited to the bi-objective case, which was instantiated on the Assignment problem with three objectives and made it possible to solve large instances.

Approximation of the efficient set Given that, on the one hand the efficient set may be hard to exploit due to its size and costly to obtain, on the other hand a single solution poorly

describes the possible outcomes of the problem, a third approach consists in generating a *representation* or an *approximation* of the efficient set. Ideally, one would like that the representation (1) covers each efficient solution, (2) consists of solutions whose outcomes are well distributed, (3) contains few solutions, and (4) is quickly obtained. Many approaches based on meta-heuristics and genetic algorithms try to satisfy all these requirements, yet without a priori guarantee on any of them. Approximation algorithms on the other hand which introduce a tolerance in the dominance relation, prove that it is possible to satisfy at least (1) and (4) – see Papadimitriou and Yannakakis (2000) –, and also show to what extent it is possible or not to also satisfy (3) – see Diakonikolas and Yannakakis (2007) and Vassilvitskii and Yannakakis (2005) –. Although primarily of theoretical interest only, it was followed by practical efficient algorithms (e.g. Bazgan et al. (2009a))

MOCO today Today, research in MOCO is very active and is supported by several grants. The work in this thesis was partially supported by the French ANR project GUEPARD (GUaranteed Efficiency for PAReto optimal solutions Determination) which creates synergy among three research centers (Lamsade, Université Paris-Dauphine; LINA, Université de Nantes; and LIP6, Université Pierre et Marie Cuirie, Paris) on all these issues.

Scope and goal of the thesis

This thesis deals mainly with the exact solution of MOCO problems, in the sense of the generation of a reduced efficient set. We are interested in approaches not limited to the bi-objective case. The general multi-objective case raises several difficulties, one of them being the computation of an explicit representation of the search region. The search region corresponds, given a set of feasible solutions to a MOO problem, to a part of the objective space where the images of the remaining efficient solutions are guaranteed to lie. We make new proposals for computing this representation and discuss related complexity aspects. We also propose a new practical efficient approach for the generation of the reduced efficient set of a MOCO problem. Our method is general in the sense that it only relies on the possibility to use a ranking algorithm on the single-objective version of the MOCO problem. Following Bazgan et al. (2009a) who proposed a very efficient algorithm to approximate, with a priori guarantee, the multi-objective Knapsack problem also supported by computational experiments, we show that our method can be adapted to efficiently approximate the reduced efficient set with a priori guarantee. The proposed approach is instantiated on the Minimum Spanning Tree problem which has numerous applications in the single objective case and presents an interesting combinatorial structure, which is exploited to specialize the proposed general method.

Outline of the thesis The first chapter is devoted to the presentation of useful concepts in MOO and the main generic methods encountered in the literature to solve MOCO prob-

lems. The second chapter deals with the multi-objective version of the Minimum Spanning Tree problem. The third chapter elaborates on the explicit representation of the search region in MOO. In the last chapter, we introduce a general MOCO solving method and present some computational experiments. Some conclusions and perspectives are finally provided.

Chapter 1

Concepts and generic algorithms in multi-objective combinatorial optimization

This first chapter introduces multi-objective optimization (MOO). The first part contains definition of several useful concepts in multi-objective optimization as well as some well known results in the subject. We define the problem setting, usual binary relations to compare solutions and points, the weighted sum scalarization technique and its relation to Pareto dominance and optimality. The first part finishes with a brief survey of solution concepts in MOO. The second part is concerned with a survey of some known generic solving methods in multi-objective combinatorial optimization. We present multi-objective dynamic programming and branch and bound on the one hand, and methods based on solving several budget constrained programs and two-phase methods on the other hand.

Contents

1.1 Preliminary concepts . . . 13
1.1.1 Problem setting . . . 13
1.1.2 Preference relations in multi-objective optimization . . . 14
Pareto dominance • Approximate dominance
1.1.3 Nondominance and efficiency . . . 15
Definitions related to elements • Definitions related to sets • Ideal and nadir points
1.1.4 Weighted sum scalarization . . . 17
Weighted sum comparisons and dominance • Weighted sum optimality and non-dominance • Supported and nondominated extreme solutions and points
1.1.5 Solution approaches . . . 18
Exact solution • Approximation with a priori guaranteed quality
1.2 Implicit enumeration in MOCO . . . 19
1.2.1 Overview . . . 19
Notations and concepts • Principle • Pairwise comparison of search nodes • Comparison of a search node to a set of solutions
1.2.2 Towards multi-objective branch and bound . . . 21
Approximation of the set of extensions of a search node in the objective space • Construction of a convex linear approximation of $Y(s)$
1.3 Exploration of the objective space . . . 25
1.3.1 Methods based on the resolution of budget constrained MOCO problems 25
1.3.2 Two-phase methods . . . 27
Phase one • Phase two

Introduction

Several practically efficient algorithms have been proposed from the end of the seventies to enumerate efficient solutions in multi-objective combinatorial optimization (MOCO). Most proposals first addressed the bi-objective case which has interesting and exclusive properties compared to higher dimensional problems. The generalization of these algorithms to more than two objective functions started to produce good practical results during the last decade. It turns out that most exact multi-objective algorithms fall into one of the two following categories.

The first category corresponds to implicit enumeration, that is, iterative partitioning of the instance and elimination of dominated subinstances. Thus this kind of algorithm primarily acts in the decision space. Two main approaches are concerned: multi-objective branch and bound (MOBB) and dynamic programming, see e.g. respectively Jorge (2010) and Bazgan et al. (2009b) for the implementation of these two methods in the general multi-objective case.

The second category contains algorithms that primarily act in the objective space, attempting to identify small zones that are individually explored. The two-phase method (see Ulungu and Teghem (1995) for the original method and Przybylski et al. (2010b) for a generalization to more than two objective functions) falls into this category as well as several methods based on the resolution of budget constrained integer programs that use the integer programming formulation of the underlying problem (see e.g. Sylva and Crema, 2004).

We present in Section 1.1 general concepts in MOO. Section 1.2 deals with implicit enumeration in MOCO. Section 1.3 surveys generic solving methods which rely on the division of the objective space. Finally, we summarize the chapter and present some topical issues in the field.

1.1 Preliminary concepts

This section defines usual concepts in multi-objective optimization and discusses solution approaches in the field.

1.1.1 Problem setting

We consider multi-objective optimization (MOO) problems

$$\begin{cases} \min & c(x) = (c_1(x), \ldots, c_p(x)) \\ \text{s.t.} & x \in X \end{cases}$$

where $X \neq \emptyset$ stands for the set of feasible solutions or *feasible set* that defines the possible choices and $c_1, \ldots, c_p$ are p *criterion*, *cost* or simply *objective* functions. We assume in this thesis that X is implicitly defined by a set of constraints and we cannot reasonably enumerate all elements of X. In multi-objective *combinatorial* optimization, X is discrete, which is the case of many problems arising in operations research.

We denote by $\mathcal{X} \supseteq X$ the *decision space*, a set of m-dimensional vectors. In MOO $\mathcal{X}$ is $\mathbb{R}^m$ (the m product set $\mathbb{R}\times\cdots\times\mathbb{R}$) or some restriction to positive and/or integral component values. In MOCO, it is generally a product set of binary values, that is, $\mathcal{X} = \{0,1\}^m$.

Many problems arising in operations research consist in selecting subsets of edges of a graph (e.g. Minimum Spanning Tree, Shortest Path, Maximum Matching), vertices of a graph (Minimum Dominating Set), or simply items out of a set (Knapsack). Assuming the set of items that can be selected is E, with $|E| = m$, the decision space can be represented equivalently by 2^E or by the set of all indicator vectors of 2^E, $\{0,1\}^m$.

An objective function c_j can take several functional forms: additive, i.e. $c_j(x) = \sum_{i=1}^{m} c_j(x_i)$, bottleneck, i.e. $c_j(x) = \max_{i=1}^{m} c_j(x_i)$ or $c_j(x) = \min_{i=1}^{m} c_j(x_i)$, or even capture some structural properties of a solution (e.g. number of leaves of a spanning tree). We assume that they map elements of $\mathcal{X}$ to $\mathbb{R}$. The vector function $c = (c_1, \ldots, c_p)$ hence maps decision vectors to *points* of the objective space $\mathcal{Z} \subset \mathbb{R}^p$. $Y = c(X)$ is the set of all feasible points.

1.1.2 Preference relations in multi-objective optimization

Pareto dominance

Points and solutions are basically compared using the concept of Pareto dominance.

For any $z, z' \in \mathcal{Z}$, we define the following relations:

$$
\begin{array}{lll}
z = z' & \text{iff} & z_j = z'_j, \text{ for all } j \in \{1,\ldots,p\} \\
z \leqq z' \quad (z \textit{ weakly dominates } z') & \text{iff} & z_j \leq z'_j, \text{ for all } j \in \{1,\ldots,p\} \\
z < z' \quad (z \textit{ strictly dominates } z') & \text{iff} & z_j < z'_j, \text{ for all } j \in \{1,\ldots,p\} \\
z \leq z' \quad (z \textit{ dominates } z') & \text{iff} & z \leqq z' \text{ and } z' \neq z
\end{array}
$$

We define in the same way the relations $\geqq$, $>$ and $\geq$.
These definitions are propagated in the decision space as follows, for any $x, x' \in \mathcal{X}$:

$$
\begin{array}{lll}
x \approx x' & \text{iff} & c(x) = c(x') \\
x \precapprox x' \quad (x \textit{ weakly dominates } x') & \text{iff} & c(x) \leqq c(x') \\
x \prec x' \quad (x \textit{ strictly dominates } x') & \text{iff} & c(x) < c(x') \\
x \preccurlyeq x' \quad (x \textit{ dominates } x') & \text{iff} & c(x) \leq c(x')
\end{array}
$$

Approximate dominance

We can introduce a tolerance in the above dominance relations and define therefore approximate dominance relations. This can be relevant for example if, from the context, small differences between objective values are judged non significant.

There are several ways to take into account a tolerance in a comparison between two points (see e.g. White, 1986). In particular, it is possible to introduce an additive term ε or a multiplicative factor $1+\varepsilon$. The latter is interesting from the complexity point of view, as shown in Papadimitriou and Yannakakis (2000). Therefore, the tolerance is assumed to be modeled by a multiplicative factor in the remainder.

For any $z, z' \in \mathcal{Z}$ and given $\varepsilon > 0$, we define the following relation:

$$\begin{array}{c} z \leqq_\varepsilon z' \\ (z \text{ } \varepsilon\text{-dominates or } \varepsilon\text{-covers } z') \end{array} \left| \begin{array}{l} \\ \text{iff } z_j \leqq (1+\varepsilon) \cdot z'_j \text{ for all } j \in \{1, \ldots, p\} \end{array} \right.$$

This relation is also translated into the decision space, for any $x, x' \in \mathcal{X}$:

$$x \precapprox_\varepsilon x' \text{ iff } c(x) \leqq_\varepsilon c(x')$$

For the relations defined above, if for two objective values $a, b \in \mathbb{R}$ the ratio $\frac{a}{b}$ is less than or equal to $1+\varepsilon$ for some tolerance $\varepsilon > 0$, then a and b are considered to be approximately identical outcomes.

1.1.3 Nondominance and efficiency

Definitions related to elements

The previous binary relations allow us to define the following concepts related to points and solutions.
Let $z \in Y$ be a point.

- z is said to be *nondominated* if there is no $z' \in Y$ such that $z' \leq z$, otherwise, it is *dominated* ;
- z is said to be *weakly nondominated* if there is no $z' \in Y$ such that $z' < z$, otherwise, it is *strictly dominated*.

A weakly nondominated point is thus either a nondominated point or a dominated point which is not strictly dominated.

Nondominated points are also called *minima* since they are minimal elements for the relation $\leq$. This term is preferred to "nondominated" in the field of computational geometry

(see e.g. Preparata and Shamos, 1985). Nondominated points are also often equivalently called *Pareto optima*.
The corresponding concepts are defined in the decision space. Let $x \in X$ be a feasible solution.

- x is said to be *efficient* if there is no $x' \in X$ such that $x' \precsim x$
- x is said to be *weakly efficient* if there is no $x' \in X$ such that $x' \prec x$

We may also extend nondominance to subsets of Y. Let $Q \subseteq Y$ and $z \in Q$.

- z is said to be *nondominated w.r.t.* Q if there is no $z' \in Q$ such that $z' \leq z$
- z is said to be *weakly nondominated w.r.t.* Q if there is no $z' \in Q$ such that $z' < z$

Definitions related to sets

According to the previous concepts on elements of Y and X, we define the following sets:

- Y_{nd} is the set of all nondominated points or simply *the nondominated set*. It is also known as the *Pareto front* or even the *skyline* in the field of databases.
- Y_{wnd} is the set of all weakly nondominated points
- X_{eff} is the set of all *efficient* solutions or simply *the efficient set*.
- X_{weff} is the set of all weakly efficient solutions

For any $Q \subseteq Y$, we also define the following sets:

- Q_{nd} is the set of all *nondominated* points *w.r.t.* Q
- Q_{wnd} is the set of all *weakly nondominated* points *w.r.t.* Q

Finally, a subset N of $\mathcal{Z}$ is *stable* for some binary relation $S_{\mathcal{Z}} \subset \mathcal{Z} \times \mathcal{Z}$ if we have

$$\text{for all } z, z' \in N, \text{ not}(z S_{\mathcal{Z}} z')$$

In the remainder, if nothing else is said, "stable" is related to the dominance relation $\leq$, and N denote a stable set with respect to this relation.

Ideal and nadir points

Denote respectively by z_j^{I} and z_j^{N} the minimal and maximal values of any point in Y_{nd} on component $j \in \{1, \ldots, p\}$. Then the points $z^{\text{I}} = (z_j^{\text{I}})_{j \in \{1,\ldots,p\}}$ and $z^{\text{N}} = (z_j^{\text{N}})_{j \in \{1,\ldots,p\}}$ are respectively called *ideal* and *nadir* points of the underlying MOO problem and instance.

1.1.4 Weighted sum scalarization

Let $\lambda = (\lambda_1, \ldots, \lambda_p) \in \mathbb{R}^p_{\geq}$ be a weight ector. We define the *weighted sum scalarization* of the objective functions $c_1, \ldots, c_p$ according to the *weight vector* λ as the function $c_\lambda = \sum_{j=1}^{p} \lambda_j c_j$

We recall below some well-known and useful results. They can be found e.g. in Ehrgott (2005).

Weighted sum comparisons and dominance

There are one way implications between dominance relations and weighted sum scalarizations of points of $\mathcal{Z}$. They are summarized in the following proposition.

Proposition 1.1. *Let z and z' be two points of $\mathcal{Z}$.*

- *If $z \leqq z'$ and $\lambda \in \mathbb{R}^p_{\geq}$, then $\lambda \cdot z \leqq \lambda \cdot z'$*
- *If $z \leq z'$ and $\lambda \in \mathbb{R}^p_{\geq}$, then $\lambda \cdot z \leqq \lambda \cdot z'$*
- *If $z \leq z'$ and $\lambda \in \mathbb{R}^p_{>}$, then $\lambda \cdot z < \lambda \cdot z'$*
- *If $z < z'$ and $\lambda \in \mathbb{R}^p_{\geq}$, then $\lambda \cdot z < \lambda \cdot z'$*

Weighted sum optimality and nondominance

Now let $z^{*\lambda}$ be the image in the objective space of an optimal solution to $\min\{c_\lambda(x) : x \in X\}$. We have the following well known results:

Proposition 1.2. *Assume $z^{*\lambda} = c(x^*_\lambda)$ where $x^*_\lambda \in \arg\min\{c_\lambda(x) : x \in X\}$.*

- *If $\lambda \in \mathbb{R}^p_{>}$, then $z^{*\lambda} \in Y_{\text{nd}}$*
- *If $\lambda \in \mathbb{R}^p_{\geq}$, then $z^{*\lambda} \in Y_{\text{wnd}}$*

Supported and nondominated extreme solutions and points

We define in this section some useful concepts related to convexity in the objective space. Some of them are illustrated in Figure 1.1.

For any $Q \subset \mathcal{Z}$, $\text{conv}(Q)$ denotes the convex hull of Q. A facet of a polytope is called *nondominated* if any normal vector λ to its supporting hyperplane directed towards the interior of the polytope has only strictly positive component values (we recall that we consider minimization problems). Otherwise, if $\lambda \geq \mathbf{0}$, then the facet is called *weakly nondominated*. Otherwise, it is *dominated*. Such a normal vector is referred to as a weight vector associated with the corresponding facet.

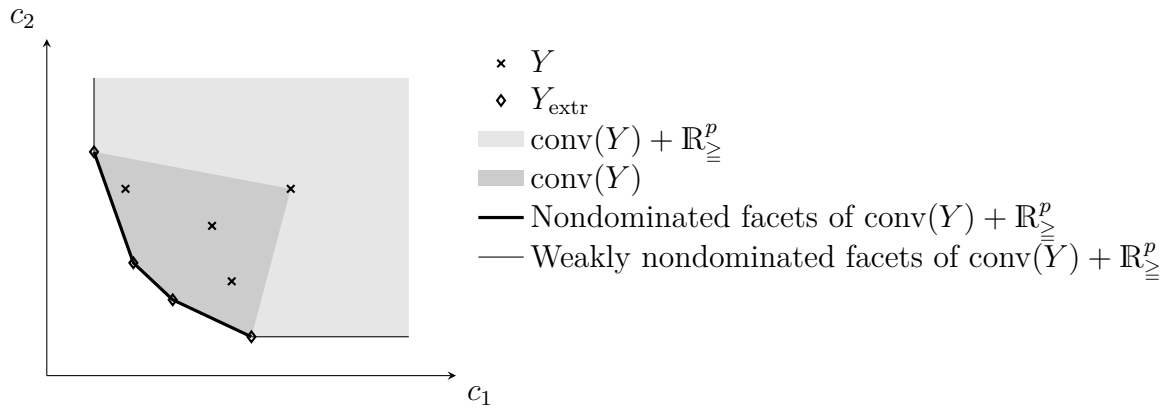


Figure 1.1: Convexity concepts in MOO

Consider the convex hull $\text{conv}(Y)$ of the set of all feasible points of a multi-objective optimization problem. The nondominated points lying on the boundary of $\text{conv}(Y)$, as well as any solution that achieves such an outcome, are called *supported*. Equivalently, a supported solution is an optimal solution to a weighted sum scalarization of the objective functions with positive weights. We denote respectively by Y_{supp} and X_{supp} the sets of all supported points and solutions.

Nondominated extreme points of $\text{conv}(Y)$ are called *extreme nondominated* or *extreme supported* points. *Extreme efficient* or *extreme supported* solutions are defined similarly to supported solutions. They are respectively denoted by Y_{extr} and X_{extr}.

Finally, for any $Q \subseteq \mathcal{Z}$, we define the set $\text{conv}(Q)_{\geqq} = \text{conv}(Q) + \mathbb{R}^p_{\geqq} = \{z + z' : z \in \text{conv}(Q), z' \in \mathbb{R}^p_{\geqq}\}$. Note that all facets of $\text{conv}(Q)_{\geqq}$ are either nondominated or weakly nondominated.

1.1.5 Solution approaches

This section presents the two solution approaches to MOO problems that are considered in this thesis.

Exact solution

Solving a multi-objective optimization problem in an exact way means that we want to obtain either the whole efficient set X_{eff} or a subset of X_{eff} containing exactly one solution per nondominated point, which we call *a reduced efficient set*. Hansen (1980) calls the former *maximal complete set* and the latter *minimal complete set*.

Algorithms in the literature devoted to multi-objective optimization generally only compute a reduced efficient set. This will also be our interpretation of the exact solution of a multi-objective optimization problem in the remainder.

Approximation with a priori guaranteed quality

Using approximate dominance relations, it is possible to define sets of solutions or points which approximate the efficient set or the non dominated set.
Given a tolerance $\varepsilon > 0$, we define the following notions:

- Y_ε is an *approximate nondominated set* if for any $z' \in Y$ there exists a $z \in Y_\varepsilon$ such that $z \leqq_\varepsilon z'$. If for any $z, z' \in Y_\varepsilon$, $z \not\leq z'$ then Y_ε is a *reduced* approximate nondominated set.
- A $(1+\varepsilon)$-*approximation* X_ε is any subset of X whose image in the objective space is an approximate nondominated set for the same tolerance. If for any $x, x' \in X_\varepsilon$, $x \not\approx x'$ then X_ε is a reduced $(1+\varepsilon)$-approximation.

As in the exact approach, approximation algorithms usually generate a reduced $(1+\varepsilon)$-approximation.

1.2 Implicit enumeration in multi-objective combinatorial optimization

Implicit enumeration algorithms operate by iteratively partitioning the set X of all feasible solutions of an optimization problem and removing elements of the partition that yield non desirable solutions. They lie in the *divide and conquer* paradigm.

We first present an overview of implicit enumeration in MOCO, then we detail a particular approach, namely multi-objective branch and bound (MOBB).

1.2.1 Overview

The process of iteratively partitioning the set X of all feasible solutions can be represented by a rooted search tree. The search tree, which initially consists of the root representing X, grows during the iterative process. A node of the search tree in general represents a subset of X and sibling nodes represent a partition of the set of solutions associated to their parent node.

The search tree is an abstract representation of the enumeration scheme. What needs to be maintained is the list of pending nodes of the search tree.

Notations and concepts

Let r denote the root of the search tree. Each search node s represents a subset of X defined by assignments to 0 or 1 of some of the variables. $X(s) \subset X$ denotes the set of feasible solutions of s, each of which is called an *extension* of s. Note that $X(r) = X$. $Y(s) = c(X(s))$ is the set of the images in the objective space of the extensions of s.

Principle

At each iteration of the procedure, a pending search node is selected. It may be divided into children nodes according to some partition scheme or discarded if it has no desirable extensions.

If the goal is to compute the efficient set, an extension is desirable if it is efficient. If just a reduced efficient set is required, an extension is desirable if it is efficient and no solution achieving the same outcome has been computed so far or could be provably obtained. As in this thesis in general, we focus on the latter. Therefore, in this case, the privileged relation in the comparison of solutions and points is the weak dominance relation, i.e. respectively $\precnsim$ and $\leqq$.

Since it is often NP-complete to determine if a given extension is efficient, the idea is to compare extensions of a search node to extensions of other search nodes or other solutions.

Namely, in order to prune a search node s, one or both of the following methods can be instantiated depending on how the exploration of the search tree is conducted:

1. find another search node s' such that for each extension of s, there is at least one extension of s' which weakly dominates it,

2. show that any extension of s is weakly dominated by some already computed solution.

We now detail these methods. Note that since we are not considering the structure of the feasible solutions, the description will be conducted in the objective space.

Pairwise comparison of search nodes

The first method can be viewed as a pairwise comparison of search nodes. It is also known[1] as a *dominance relation*. Formally, it is defined as follows, for any search nodes s and s':

$$s' \text{ dominates } s \text{ iff for all } z \in Y(s), \text{ there exists } z' \in Y(s') \text{ such that } z' \leqq z$$

Such dominance relations are efficiently used in multi-objective dynamic programming, which is a kind of implicit enumeration. Actually, in Bazgan et al. (2009b) several dominance relations are used at each stage of a dynamic programming algorithm to compute a reduced efficient set of the multi-objective KNAPSACK problem. The computational experiments they conducted even show that their dominance relations are complementary.

Comparison of a search node to a set of solutions

We may, at some point of the enumeration, have computed a set of solutions P. The set P, or its image $N = c(P)$ in the objective space, may be generated before the enumeration through

[1] not only in MOO, see e.g. Jouglet and Carlier (2011) for a survey on the use of such pairwise comparisons in combinatorial optimization, especially for scheduling problems

a procedure dedicated to the computation of an initial set of (non necessarily efficient) solutions. It may also contain some solutions obtained during the previous enumeration steps, if the search tree is explored following some kind of depth first or best first order. In both cases, the points of N correspond to feasible points, thus any point dominated by a point of N needs not be kept in the output. This gives rise to the following pruning rule:

$$\text{prune } s \text{ if for all } z \in Y(s), \text{ there exists } z' \in N \text{ such that } z' \leqq z \tag{1.1}$$

1.2.2 Towards multi-objective branch and bound

We consider in this section a particular implicit enumeration scheme which basically relies only on the second pruning method. Therefore we first explain how a convex linear approximation of the set of extensions of a search node can be used in a pruning rule, then how such an approximation can be obtained.

Approximation of the set of extensions of a search node in the objective space

In the previous section, the pruning rule (1.1) uses a stable set of already computed feasible points to circumvent the fact that neither X_{eff} nor Y_{nd} are known. However, the general pruning rules we presented also involve the extensions of a search node s which is not known either. Hence, there is a need to approximate $Y(s)$. More precisely, we discuss how a sub-space $L(s)$ containing $Y(s)$ can be determined.

The easiest way to proceed is to compute for each objective j a lower bound on the value of any extension of s, namely a value $l_j(s)$ such that for any $z \in Y(s)$, $l_j(s) \leq z_j$. Thus the set $L(s) = \{z \in \mathcal{Z} : l(s) \leqq z\}$ is a correct approximation of $Y(s)$ (i.e. $L(s) \supset Y(s)$) and the following pruning rule directly follows:

$$\text{prune } s \text{ if there exists } z' \in N \text{ such that } z' \leqq l(s)$$

For MOCO problems whose single-objective version is solvable in polynomial time, the best lower bound for objective c_j is obtained by computing a point $z^{*j}(s)$ which is optimal for objective c_j for the sub-instance s, i.e.

$$z^{*j}(s) = \arg\min\{z_j : z \in Y(s)\}.$$

Therefore, we take $l(s) = (z_1^{*1}(s), \ldots, z_p^{*p}(s))$.

For MOCO problems whose single-objective version is NP-hard, the computation of single objective optima may be expensive and it is possible to rely on non optimal lower bounds, such as the optimal value of the linear relaxation of the underlying problem. For example, Bazgan et al. (2009b) use a lower bound computed in linear time for the single-objective Knapsack problem to implement a pruning relation based on this principle denoted there

by D_b^k.

The single point evaluation is coarse in practice. In Figure 1.2a, we illustrate this with an instance where all extensions of a search node s are dominated, while the approximation of $Y(s)$ through the single point $l(s) = (z_1^{*1}(s), z_2^{*2}(s))$ intersects the search region. Thus in this example, the pruning rule is inefficient, and we see that a refined approximation of $Y(s)$ would be profitable.

The previous approximation based on a single point is a particular case of approximation based on the weighted sum optimization technique. Using any discrete set $\Lambda \subset \mathbb{R}^p_{\geq}$ of weight vectors it is possible to obtain the following approximation of $Y(s)$:

$$L_\Lambda(s) = \{z \in \mathcal{Z} : \forall \lambda \in \Lambda, \lambda \cdot z^{*\lambda}(s) \leqq \lambda \cdot z\}$$

where $z^{*\lambda}(s)$ is an optimal point for $\min\{\lambda \cdot z : z \in Y(s)\}$.

We have $L_\Lambda(s) \supseteq Y(s)$. Therefore, $L_\Lambda(s)$ can be used in place of $Y(s)$ in Equation (1.1). However $L(s)$ is not a discrete set, which implies that the pruning rule formulated as in Equation (1.1) cannot be tested. We shall see in Chapter 3 how local upper bounds provide an alternative formulation of this rule, which is suitable for practical testing.

A simple situation in the bi-objective case is illustrated in Figure 1.2b, where $Y(s)$ is approximated through the optima of s for objective functions c_1, c_2 and c_λ with $\lambda = (1, 1)$. In this example, s cannot yield any new non dominated point. Therefore it can be pruned.

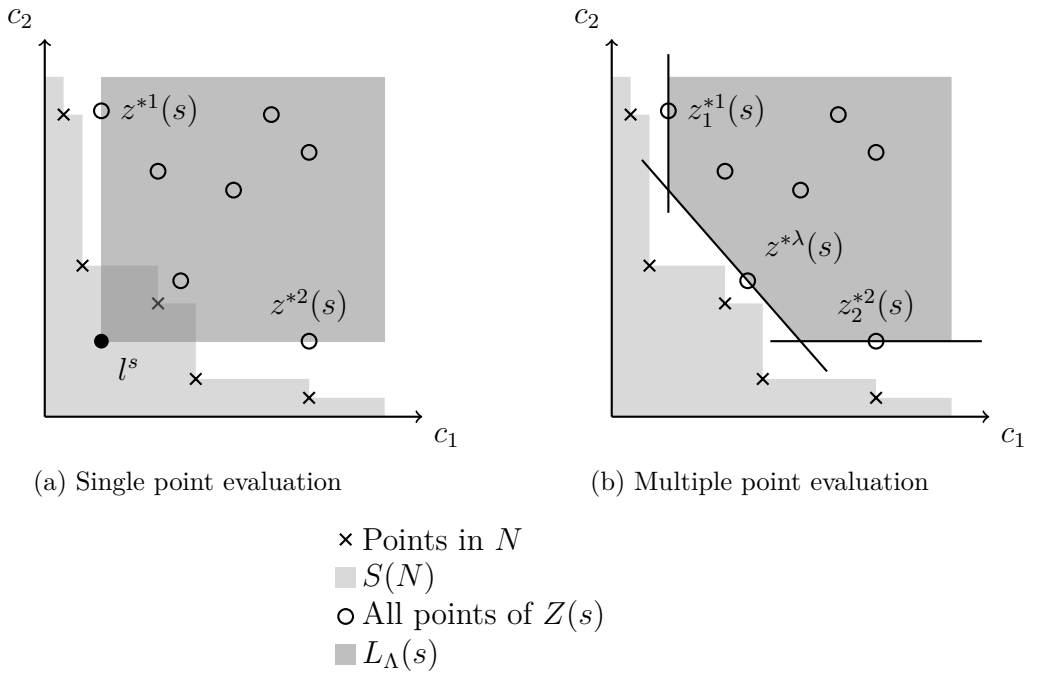
The use of pruning rules based on the approximation of $Y(s)$ is the foundation of *multi-objective branch and bound*. It seems to have first been introduced by Villarreal and Karwan (1981) to solve MOCO problems formulated as integer programs, especially the KNAPSACK problem. Sourd and Spanjaard (2008) enumerate all extreme points of any search node s in order to compute $\text{conv}(Y(s))_{\leqq}$ as an approximation of $Y(s)$.

Construction of a convex linear approximation of $Y(s)$

Basically, any set of weight vectors from $\mathbb{R}^p_{\geq}$ can be used to approximate $Y(s)$. The quality of the approximation, however, depends on the choice of weight vectors.

The set of weight vectors that yields the best approximation of $Y(s)$ is the set of supporting hyperplanes of $\text{conv}(Y(s))_{\leqq}$. This is closely related to the computation of the set Y_{extr} of all extreme supported points, which are the extreme points of $\text{conv}(Y(s))_{\leqq}$. When the single objective version of the MOCO problem is easily solved, Y_{extr} can be efficiently computed in the bi-objective case using the dichotomic algorithm of Aneja and Nair (1979), which achieves this goal requiring $O(|Y_{\text{extr}}|)$ resolutions of the single-objective version.

The method of Aneja and Nair (1979) is described in Algorithm 1.1. A set of weight vectors associated with each nondominated facet of $\text{conv}(Y)_{\leqq}$ may be generated at the same time, therefore, the required instructions are integrated to Algorithm 1.1. In order to ensure that the points obtained as optima of weighted sum objectives are nondominated and



(a) Single point evaluation

(b) Multiple point evaluation

Figure 1.2: Comparison of evaluations of $Y(s)$ in the bi-objective case

extreme, lexicographic optimization is used. For example, at the second step, we compute a point which is optimal for the weight vector λ^2 among all optima according to λ^1.

For instance, the MOBB algorithm of Sourd and Spanjaard (2008) to solve the bi-objective Minimum Spanning Tree problem relies on this dichotomic scheme.

The general multi-objective case raises some difficulties in the generalization of the dichotomic scheme presented above. A first difficulty, for problems with more than two objectives, is the possible existence of several nondominated points optimizing the same objective function. Moreover, even if it is not the case, the hyperplane passing through the p optimal points found during the initialization may be associated to a weight vector with entries of opposite signs, the optimization of which may return a dominated point. Przybylski et al. (2010a), who pointed out these issues, got around them by working in the weight vector space defined as $W^0 = \{\lambda \in \mathbb{R}^p_{\geqq} : \lambda_p = \sum_{j=1}^{p-1} \lambda_j\}$. Their method is *recursive* in the sense that in order to output the set Y_{extr} of a p-objective problem, the sets Y_{extr} of several $(p-1)$-objective problems have to be computed. We note that Özpeynirci and Köksalan (2010) proposed a non recursive generalization of the dichotomic scheme. Their approach relies on the use of p dummy points together with the at most $p!$ nondominated points optimizing separately all objectives, which roughly makes it possible to quickly adapt the bi-objective scheme. The problem of hyperplanes defining non convex linear combinations is bypassed by discarding the associated weight vectors. Moreover, dummy points are used

Algorithm 1.1: Computation of the set Y_{extr} in the bi-objective case (Aneja and Nair (1979))

$\lambda^1 \leftarrow (1,0)$; $\lambda^2 \leftarrow (0,1)$

$z^{*1} \leftarrow \arg\operatorname{lexmin}\{(\lambda^1 \cdot z, \lambda^2) \cdot z : z \in Y\}$; $z^{*2} \leftarrow \arg\operatorname{lexmin}\{(\lambda^2 \cdot z, \lambda^1) \cdot z : z \in Y\}$
– *The lexicographic objective avoids weakly dominated points.*

$O \leftarrow \{(z^{*1}, z^{*2})\}$ – *List of open pairs of extreme points to exploit*

$Y'_{\text{extr}} \leftarrow \{z^{*1}, z^{*2}\}$ – *Initialization of the output*
– *which will eventually contain all points of* Y_{extr}.

$\Lambda \leftarrow \{\lambda^1, \lambda^2\}$ – *Initialize a set which will eventually contain*
– *one weight vector associated to each nondominated facet of* $\operatorname{conv}(Y)_{\leqq}$

while $O \neq \emptyset$ **do**

  Select $(z^1, z^2) \in O$; $O \leftarrow O \setminus \{(z^1, z^2)\}$

  $\lambda \leftarrow (|z_2^1 - z_2^2|, |z_1^1 - z_1^2|)$ – *Compute a normal to the straight line* (z^1, z^2).

  $z^{*\lambda} \leftarrow \arg\operatorname{lexmin}\{(\lambda \cdot z, \lambda^1 \cdot z) : z \in Y\}$ – *The lexicographic objective avoids*
  – *non extreme points.*

  if $\lambda \cdot z^{*\lambda} < \lambda \cdot z^1$ **then**

    $O \leftarrow O \cup \{(z^1, z^{*\lambda}), (z^{*\lambda}, z^2)\}$

    $Y'_{\text{extr}} \leftarrow Y'_{\text{extr}} \cup \{z^{*\lambda}\}$

  else

    $\Lambda \leftarrow \Lambda \cup \{\lambda\}$ – λ *is a weight vector associated*
    – *to a nondominated facet of* $\operatorname{conv}(Y)_{\leqq}$, *namely* $[z^1, z^2]$

return $Y_{\text{extr}} = Y'_{\text{extr}}$, Λ

to ensure that the set Y_{extr} is obtained at the end. The method seems to be practically less efficient than that of Przybylski et al. (2010a).

However, in Przybylski et al. (2010a) and Özpeynirci and Köksalan (2010), only the tri-objective case is solved within a reasonable amount of time.

1.3 Exploration of the objective space

In MOCO, not all exact solving methods follow implicit enumeration. Instead, many exact methods primarily operate in the objective space by searching in several, often overlapping, subspaces. We first mention methods primarily relying on the resolution of several integer programs. Then we review the two-phase method since it was instantiated on many MOCO problems and was subject to several enhancements and extensions. In the latter, we put an emphasis on the ranking strategy in the second phase, since it is one of the ingredients of the hybrid approach we propose in Chapter 4.

1.3.1 Methods based on the resolution of budget constrained MOCO problems

Several works propose to find all nondominated points of a MOCO problem by solving integer programs that comprise constraints that define the feasible set of the underlying problem plus additional constraints. The principle is to formulate an integer program that excludes from the feasible set all the solutions that are dominated by at least one of the previously computed solutions. Therefore, all nondominated points are found after solving a series of integer programs.

These integer programs are often derived from budget constrained versions of the initial problem formulated as an integer program. This usually consists in turning all but one objective into *budget*, resource, or knapsack constraints while optimizing with respect to the remaining one, e.g.

$$\begin{cases} \min & c_1(x) \\ \text{s.t.} & x \in X \\ & c_j(x) \leq b_j \quad j = 2, \dots, p \end{cases} \tag{1.2}$$

for some $b_2, \dots, b_p \in \mathbb{R}$. The solution to such problems is often significantly harder than the unconstrained version. Therefore, for several classical operations research problems, specialized exact and approximation algorithms have been developed[2]. As we will see, the computation of values for the parameters $b_2, \dots, b_p$ was first integrated into (1.2), adding even more constraints and binary variables.

[2]See e.g. Aboudi and Jørnsten (1990) and Aggarwal (1985) for the Assignment problem, Aggarwal et al. (1982), Hassin and Levin (2004), and Ravi and Goemans (1996) for the bi-objective and Grandoni et al. (2009) and Hong et al. (2004) for the multi-objective Minimum Spanning Tree problem.

In Sylva and Crema (2004, 2007), a single program looks for a point in a search region defined according to the points found so far, which excludes the part of the objective space that these points dominate. If the program has no feasible solution, then the search region contains no feasible point, which ensures that all nondominated points have been found. Besides, the objective function is defined so as to obtain efficient optimal solutions (they made several proposals for this). In their approach, each new point z induces new constraints that are added to the program to ensure that future points improve on at least one objective with respect to z. This is modelled through a conjunction of disjunctive constraints. The linearization of the disjunctive constraints induces big-M coefficients and additional binary variables, therefore solving the program becomes rapidly very expensive.

The approach was later enhanced in Sylva and Crema (2008). In this paper, they split the integer programs formulated in their earlier works into two integer programs. The first program determines for each previously computed solution which objective a new solution should improve. The solution to this first stage program is used to formulate the second stage program, which is just a budget constrained version of the integer programming formulation of the underlying problem. Contrary to the former approaches, the second stage program looks for points in a single search zone determined after solving the first stage program, rather than in the whole search region. This is basically the same approach as the first application of local upper bounds we present in Section 3.1.5. However, Sylva and Crema (2008) solve integer programs to compute these local upper bounds (namely the first stage programs) whereas they can be quickly computed thanks to the algorithms we present in Chapter 3. Therefore, their second method remains costly.

Laumanns et al. (2006) also presented an approach which involves the resolution of budget constrained programs that search inside boxes. While the determination of the boxes is faster than in Sylva and Crema (2008) (it does not require to solve additional integer programs), the boxes themselves are highly redundant.

Lokman and Köksalan (2013) and Kirlik and Sayın (2014) also determine search zones to be explored by budget constrained programs and improved on earlier approaches by identifying many redundancies in the search zones before they are explored. However, the approach of Lokman and Köksalan (2013) is instantiated on the multi-objective Minimum Spanning Tree problem and solves only small instance sizes (10 vertices and 3 objective functions). We also mention that Tenfelde-Podehl (2003) and Dhaenens et al. (2010) proposed exact algorithm that recurse over the set of objective functions to partition the search space.

Other approaches rely on the use of non linear aggregation function. See e.g. Ralphs et al. (2006).

Finally, Przybylski et al. (2009) implemented the algorithms of Sylva and Crema (2004), Tenfelde-Podehl (2003), Laumanns et al. (2006), and their own two-phase method Przybylski et al. (2010b) to solve instances of the tri-objective Assignment problem. Their computational experiments showed that computation times decrease from Sylva and Crema

(2004) to Przybylski et al. (2010b).

1.3.2 Two-phase methods

The first two-phase method was introduced by Ulungu and Teghem (1995). In the first phase, the set Y_{extr} of all extreme nondominated points is computed. It is used to delimit a part of the objective space which is explored in the second phase in order to generate all non extreme nondominated points. The efficient set (or a reduced efficient set) is eventually obtained as the union of phase one and phase two solution sets.

The method was instantiated on bi-objective versions of several classical operational research problems including amongst others the Assignment problem (Przybylski et al., 2008; Delort and Spanjaard, 2011), the Knapsack problem (Visée et al., 1998; Delort and Spanjaard, 2010), the Minimum Cost Flow problem (Raith and Ehrgott, 2009), and the Minimum Spanning Tree problem (Ramos et al., 1998; Steiner and Radzik, 2008). It was also adapted to more than two objectives by Przybylski et al. (2010b) and applied to the Assignment problem and also to the Knapsack problem in Jorge (2010) (both for the case of three objectives).

We now discuss the details of the two phases.

Phase one

Phase one of the two-phase method usually consists in computing the set Y_{extr} of all extreme nondominated points (see Section 1.2.2). The remaining points all lie both in the augmented hull $\text{conv}(Y_{\text{extr}})_{\geqq}$ and in the part of the objective space that is not dominated by any point of Y_{extr}. The intersection of these sets defines a *search area*. In the bi-objective case, the search area corresponds to a union of triangles (see Figure 1.3).

Phase two

Phase two consists in the exploration of the zones defined at the end of phase one. Two exploration strategies, implicit enumeration and ranking, can be found in the literature as we can see in Table 1.1.

The first strategy consists in applying an implicit enumeration scheme (generally MOBB) to explore the zones or the whole search area.

The second strategy relies on a ranking algorithm for the single-objective version of the underlying problem. A ranking or k-best algorithm for a single-objective combinatorial optimization problem determines for a given positive number k the k best solutions in non decreasing order of their objective values. The use of ranking algorithms together with the weighted sum scalarization technique is usual in MOCO – and also recently appeared in multi-objective mixed integer programming (Vincent et al., 2011) – for two main reasons. First, it makes it possible to determine the solutions that lie in a given band in the objective

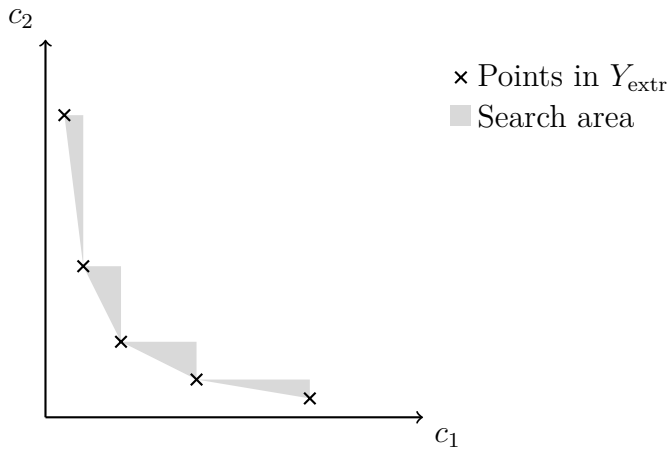


Figure 1.3: A bi-objective two-phase situation

Problem	Implicit enumeration	Ranking	Computational comparison
Minimum Spanning Tree	Ramos et al. (1998) (single point evaluation)	Steiner and Radzik (2003, 2008)	Steiner and Radzik (2008)
Assignment	Delort and Spanjaard (2011)	Przybylski et al. (2008)	Delort and Spanjaard (2011)
Minimum Cost Flow		Raith and Ehrgott (2009)	
Knapsack	Visée et al. (1998) Delort and Spanjaard (2010) Jorge (2010)	Jorge (2010)	Jorge (2010)

Table 1.1: Classification of two-phase methods in the bi-objective case according to phase two algorithm.

space whose slope is defined by the choice of a weight vector and whose width is controlled by a given maximum weighted sum value. Therefore, the choice of a weight vector to be used in the weighted sum scalarization corresponds to a *ranking direction* through the objective space. Second, general ranking algorithms as well as efficient algorithms specialized to a particular problem have been developed and used for the most studied problems in MOCO. Murty (1968) proposed a partition scheme which makes it possible to rank the solutions of any 0-1 program. But according to this scheme, it may be required to solve independently several subproblems to obtain each ranked solution. Specialized efficient algorithms include Pedersen et al. (2008) for the Assignment problem, Eppstein (1998) and Jiménez and Marzal (2003) for the Shortest Path problem, Hamacher (1995) for the Minimum Cost Flow problem, and Gabow (1977), Katoh et al. (1981), and Eppstein (1992) for the Minimum Spanning Tree problem. We present in Section 2.5 the algorithm of Katoh et al. for the Minimum Spanning Tree problem which yields the best known time complexity when the number of solutions to rank is not known in advance. Finally, note that general as well as specialized ranking algorithms often follow an implicit enumeration approach, i.e. a ranking tree.

The idea in the use of ranking algorithms in phase two is to exploit directions that correspond to weight vectors associated to the facets of $\text{conv}(Y_{\text{extr}})_{\leqq}$. It is then necessary to define a stopping condition for each run of the ranking algorithms so that the union of the bands that are explored thereby covers the search area.

As we saw, in the bi-objective case, the search area is easily partitioned into triangles whose corner points are two extreme nondominated points and a *local nadir point*. It is therefore natural to use these local nadir points to define the upper limit of the bands. The stopping condition is thus defined using the coordinates of these points.

In the general multi-objective case, such local nadir points, which we rather call *local upper bounds* in this case, are harder to define (this is an issue we address in Section 3.1.3). Moreover, local upper bounds are no longer in one-to-one correspondence with facets of $\text{conv}(Y)_{\leqq}$. Therefore there is no natural definition of zones. The approach of Przybylski et al. (2010b) addresses all these issues.

Computational comparisons of phase two strategies There have been several computational comparisons between implicit enumeration and ranking algorithms in phase two. Steiner and Radzik (2008) compared their own ranking algorithm to the implicit enumeration algorithm of Ramos et al. (1998) and concluded that their approach outperforms the latter approach. However, the approach of Ramos et al. is based on a single point evaluation model as described in Section 1.2.2. The comparisons conducted by Jorge (2010) and Delort and Spanjaard (2011) respectively on the Knapsack and Assignment problems are more uneven. In the tri-objective case, Jorge (2010) observed a large superiority of the ranking approach over both dynamic programming and multi-objective branch and bound approaches on instances whose objective values are uniformly drawn. In the bi-objective

case, the local branch and bound of Delort and Spanjaard (2011) seems to perform better than the approach of Przybylski et al. (2008), especially on low range objective scales.

Conclusions

We described in this chapter several approaches to the exact solution of MOCO problems. We classified these methods into two categories, implicit enumeration and division of the objective space, according to how the computation effort is organized.

Some trends can be drawn in the comparison of some classes of methods. Currently the methods that add (at least) budget constraints to the integer programming formulation of the underlying problem solve quite smaller instances than what MOBB and two-phase methods do. This can be observed in the computational experiments conducted by the authors of these methods. The same conclusion can be drawn from Przybylski et al. (2009). This is one of the reasons for which we were particularly interested in this thesis in multi-objective branch and bound and the use of ranking algorithms as in the second phase of the two-phase method.

Chapter 2

Some insights into the multi-objective Minimum Spanning Tree problem

This chapter deals with several aspects related to the multi-objective version of the Minimum Spanning Tree. We review computational complexity issues related to the problem and the relations between the solution to the single and the multi-objective versions. We compare the output of several approaches that generate a superset of the nondominated set. We also discuss generic MOCO solving methods that were instantiated on the Minimum Spanning Tree problem. Some of them rely on ranking spanning tree according to a single objective function; we present the efficient algorithm of Katoh et al. (1981) for this.

Contents

2.1 Notations 33
2.2 Computational hardness 34
2.2.1 Exponential number of nondominated points 34
2.2.2 NP-completeness of the associated decision problem 34
2.3 From the single objective to the multi-objective case 35
2.3.1 Cut and cycle optimality conditions 36
Cut and cycle necessary conditions • Coloring rules • Application of the coloring rules
2.3.2 Generalizations of single objective Minimum Spanning Tree algorithms 38
Preliminaries • Generalization of Prim's algorithm • Generalization of Kruskal's algorithm • Remarks
2.4 Generic MOCO solving methods applied to the MOST problem 43
2.4.1 Two-phase methods 43
2.4.2 A branch and bound algorithm 43
2.5 Ranking spanning trees 44
2.5.1 Generic ranking scheme and specialization to the Minimum Spanning Tree problem 45
2.5.2 Procedures of the ranking algorithm of Katoh et al. (1981) 46

Introduction

The generic method we present in Chapter 4 is instantiated on the multi-objective Minimum Spanning Tree (MOST) problem. The Minimum Spanning Tree is a long-standing problem in operations research (Graham and Hell, 1985; Mares, 2008) which has numerous applications: Ahuja et al. (1993, Chap. 13) give an overview of applications in the single objective case. Applications in the fields of network design and integrated circuits design often involve multiple objectives. The problem also received much attention for its solution, see e.g. Mares (2008) for a survey of algorithms.

We consider in this chapter the case of homogeneous objective functions obtained by componentwise sums of vector values associated with each edge of the graph, often referred to as *sum* objectives. Other models also consider *bottleneck* objectives, i.e. minimum or maximum of selected edges values on a given component. These approaches are surveyed from both a theoretical and an algorithmic point of view in Ruzika and Hamacher (2009).

After briefly defining some notations (Section 2.1), we recall the two main computational difficulties arising in the MOST problem (Section 2.2). Then in Section 2.3 we present several results on the single objective Minimum Spanning Tree problem and their extension to the multi-objective case. We discuss in Section 2.4 some aspects of generic MOCO solving methods we presented in the previous chapter which were applied to the MOST problem. Finally we review the algorithm of Katoh et al. (1981) for ranking spanning trees (Section 2.5).

2.1 Notations

Let $G = (V, E)$ be a connected graph whose set of vertices is V with $|V| = n$ and set of edges is E with $|E| = m$.

We are interested in the following problem:

$$\begin{cases} \min & c(t) = (c_1(t), \dots, c_p(t)) \\ \text{s.t.} & t \in T \end{cases} \tag{2.1}$$

where T is the set of spanning trees of G and $t \in T$ designates a spanning tree as well as the set of edges that constitutes it.

The decision space $\mathcal{X}$ of this problem is defined, in this thesis, as the set of all subgraphs of G, i.e. 2^E. This allows us to consider the cost of any subset of edges, in particular partial solutions corresponding to trees being constructed.

The objective or cost functions $c_1, \dots, c_p$ are assumed to be all additive, i.e. $c_j(e) = \sum_{i=1}^{m} c_j(e_i)$ for any $e \in 2^E$.

2.2 Computational hardness

Like many MOCO problems whose single objective version is solved in polynomial time, two computational difficulties emerge from the MOST problem.

2.2.1 Exponential number of nondominated points

Hamacher and Ruhe (1994) presented a bi-objective instance based on a complete graph of arbitrary many vertices, all spanning trees of which are efficient and whose objective vectors differ from one another. Since a complete graph on n vertices contains n^{n-2} spanning trees, this shows that there can be exponentially many efficient spanning trees in a MOST instance. Let K_n be a complete graph on n vertices. Assume that the edges of K_n are indexed from 1 to $m = n(n-1)/2$, namely $E = \{e_1, \ldots, e_m\}$. The cost functions are defined as follows, for any edge e_i:

$$c(e_i) = (2^{i-1}, 2^m - 2^{i-1})$$

Note that all edge costs are different and no two edges dominate each other since $c_1(e_i) + c_2(e_i)$ is a constant. From the uniqueness of the binary representation, all spanning trees costs are also different. Hence, there are as many nondominated points as efficient spanning trees. However, the ranges of the objective functions values also grow exponentially.

This property of MOCO problems is also referred to as *intractability*. It puts forward approximation algorithms which aim at outputting a smaller yet representative solution set. Especially, Diakonikolas and Yannakakis (2007), Papadimitriou and Yannakakis (2000), and Vassilvitskii and Yannakakis (2005) show that small approximation sets can be obtained for several MOCO problems.

2.2.2 NP-completeness of the associated decision problem

An interesting problem associated to MOCO is the following: given a point $z^b \in \mathcal{Z}$, does there exist a feasible point dominating z^b? This problem can be viewed as the decision version of a MOCO problem. Once more, for many MOCO problems, this decision version is NP-complete, even in the bi-objective case. This also holds for the MOST problem. This was shown by Camerini et al. (1983) through a reduction of the decision version of the bi-objective Minimum Spanning Tree from the Subset Sum problem, which asks whether among a set of values, there exists a subset of these values that sum up to a specified quantity.

Consider an instance of Subset Sum problem defined by the set of values $a_1, \ldots, a_n$ and the quantity b, all being positive integers. From this instance of Subset Sum, we construct an instance of the decision version of the BOST problem. Let G be a connected graph whose set of vertices is $V = \{1, \ldots, n+2\}$ and whose set of edges is $E = \{(i, n+1), (i, n+2)\}_{i \in \{1,\ldots,n\}} \cup \{(n+1, n+2)\}$. The cost functions are defined as follows, for

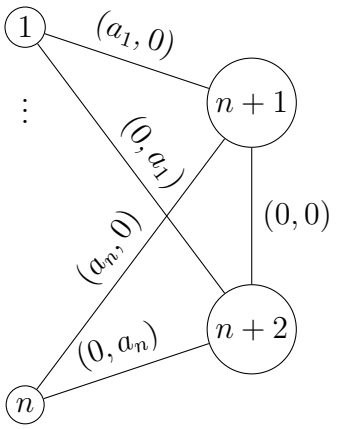


Figure 2.1: Instance of BOST constructed from any instance of Subset Sum

$i \in \{1, \ldots, n\}$:

$$\begin{aligned} c(i, n+1) &= (a_i, 0) \\ c(i, n+2) &= (0, a_i) \\ c(n+1, n+2) &= (0, 0) \end{aligned}$$

The corresponding graph is shown in Figure 2.1.

The bi-objective bound z^b is defined as $z^b = (b, \sum_{i=1}^{n} a_i - b)$. We associate to any spanning tree t of G the subset I of $\{1, \ldots, n\}$ such that $i \in I$ if $(i, n+1) \in t$ (we exclude spanning trees which do not take edge $(n+1, n+2)$ since they are dominated). Conversely, to any $I \subset \{1, \ldots, n\}$ we associate the spanning tree t whose edges are $(n+1, n+2)$ and $(i, n+1)$ for $i \in I$, $(i, n+2)$ otherwise. Hence, we have the following:

$$\begin{aligned} c_1(t) \leq z_1^b &\Leftrightarrow \sum_{i \in I} a_i \leq b \\ c_2(t) \leq z_2^b &\Leftrightarrow \sum_{i \in I} a_i \geq b \end{aligned}$$

which implies that a spanning tree t dominates z^b if, and only if, the subset I constructed as above is a solution to the Subset Sum problem.

2.3 From the single objective to the multi-objective case

We present in this section some known results related to the single objective Minimum Spanning Tree problem and discuss their extension to the multi-objective case. The

single cost function will be denoted by w, hence a single objective minimum spanning tree will be referred to as w-MST.

2.3.1 Cut and cycle optimality conditions

Given a spanning tree $t \in T$, we define the following edge sets.

Fundamental cycle For any edge $e \in E \setminus t$, $\mathrm{Cyc}(t, e)$ denotes the set of edges of the unique cycle of the subgraph $(V, t \cup \{e\})$

Fundamental cut For any edge $e \in t$, $\mathrm{Cut}(t, e)$ denotes the set of edges that reconnect the two connected components of the subgraph $(V, t \setminus \{e\})$

Cut and cycle necessary conditions

The following theorem is well known in the single objective case (see e.g. Ahuja et al., 1993, Chap. 13).

Theorem 2.1. *Let t be a w-MST. Then:*

- *for any edge $e \in t$ and for any edge $e' \in \mathrm{Cut}(t, e)$, $w(e) \leq w(e')$*
- *for any edge $e' \in E \setminus t$ and for any edge $e \in \mathrm{Cycle}(t, e')$, $w(e) \leq w(e')$*

This result can be generalized in the multi-objective case.

Theorem 2.2 (Cut and cycle necessary conditions, Hamacher and Ruhe, 1994)**.** *Let t be an efficient spanning tree. Then:*

- *any edge $e \in t$ is nondominated in $\mathrm{Cut}(t, e)$;*
- *any edge $e' \in E \setminus t$ does not dominate any edge in $\mathrm{Cycle}(t, e')$.*

A generalization of Prim's algorithm presented in the next section is based on this result. However, as we will see in the instance shown in Figure 2.3, these conditions are not sufficient to guarantee that an edge satisfying one of them belongs or not to an efficient spanning tree.

We note that Alonso et al. (2009) proposed sufficient conditions for a preference relation S over 2^E so that greedy algorithms which rely on the optimality conditions yield only minimal spanning trees for S, but these conditions are not satisfied by the dominance relation $\precsim$.

Coloring rules

In the single objective case, the well known greedy algorithms of Prim, Kruskal and Borůvka consist in the application of the two following rules.

Definition 2.3 (Coloring rules in the single objective case, Tarjan, 1983)**.** Let G be a graph with single valued edge costs. Assume that all edges of G are initially uncolored.

- (Blue rule) Let e be an edge in a cut ω of G containing no *blue* edge. If e is of minimal cost among the uncolored edges of ω, color it *blue*.
- (Red rule) Let e be an edge in a cycle ω of G containing no *red* edge. If e is of maximal cost among the uncolored edges of ω, color it *red*.

The result of the greedy application of these rules is summarized in the following theorem.

Theorem 2.4 (Tarjan, 1983)**.** *The greedy application of the above coloring rules on a connected graph yields a complete coloring of the graph such that the set of all blue edges or the complement set of all red edges defines an optimal spanning tree.*

Sourd and Spanjaard (2008) remarked that the coloring rules and the associated result immediately extend to the multi-objective case since they only assume the existence of *least* and *greatest* elements of the underlying ordering of edges in a given subset (cut or cycle).

Definition 2.5 (Coloring rules in the multi-objective case, Sourd and Spanjaard, 2008)**.** Let G be a graph with multi-valued edge costs. Assume that all edges of G are initially uncolored.

- (Blue rule) Let e be an edge in a cut ω of G containing no *blue* edge. If e is a *dominating* edge among the uncolored edges of ω (for all uncolored $e' \in \omega$, $c(e) \leqq c(e')$), color it *blue*.
- (Red rule) Let e be an edge in a cycle ω of G such that e is *dominated* by all uncolored edges of ω (for all uncolored $e' \in \omega$, $c(e') \leqq c(e)$), color it *red*.

Note that since the relation $\precapprox$ is not complete, least or greatest elements in a given subset of E may not exist. Hence in the multi-objective case, the coloring rules can (and in practice often) lead to uncolored edges. The worst case is the situation where no two edges dominate each other (that is $E = E_{\text{eff}}$), which prevents the application of any of the two rules.

However, we have the following result:

Theorem 2.6 (Sourd and Spanjaard, 2008)**.** *If the coloring rules in the multi-objective case are greedily applied on a connected graph, then for each nondominated point, there exists at least one spanning tree compatible with the coloring rules, i.e. taking all blue edges and no red edges, that achieves the same outcome.*

Application of the coloring rules

In the single objective case, the application of the coloring rules yields a complete coloring of the instance and hence a single solution.

In the multi-objective case, applying the coloring rules does not yield a single solution. Moreover, there may be dominated spanning trees among the spanning trees that satisfy the coloring (i.e. that take all blue edges and no red edge). Therefore, we can only hope a reduction in some cases of the instance. We are thus interested in applying these rules without generating all the spanning trees compatible with the colors.

From this point of view, Sourd and Spanjaard (2008) also show, as in the single objective case, that the coloring does not depend on the order these rules are applied on the edges of the graph. They provide for each rule an $O(m)$ algorithm for its application. Let blue(E) and red(E) respectively denote the current sets of blue and red edges of G and consider an uncolored edge $e = (u, v)$. In order to apply the blue rule, a depth first search is performed from u on the partial graph $G_e^{\text{cut}} = (V, E_e^{\text{cut}})$ where $E_e^{\text{cut}} = \{e' \in E \setminus \{e\} : c(e) \lneqq c(e')\} \cup \text{blue}(E)$. If v is *not* visited, e is colored blue, since the partition between visited and unvisited vertices is a cut that satisfies the conditions of the blue rule. Similarly, to apply the red rule, a depth first search is performed from u on the partial graph $G_e^{\text{cyc}} = (V, E_e^{\text{cyc}})$ where $E_e^{\text{cyc}} = \{e' \in E \setminus (\text{red}(E) \cup \{e\}) : c(e') \leqq c(e)\} \cup \text{blue}(E)$. If v is visited, e is colored red, since we traversed a path which, together with e, constitutes a cycle that satisfies the conditions of the red rule.

2.3.2 Generalizations of single objective Minimum Spanning Tree algorithms

Preliminaries

We consider in this section generalizations of Prim's and Kruskal's algorithms to the multi-objective case. We recall, for the single objective versions, that:

- Prim's algorithm grows a tree which spans a subset of the graph vertices. Therefore, a tree t is represented by a partial subgraph (V_t, E_t). The algorithm starts from a tree $t = (V_t, E_t)$ where $V_t = \{r\}$ and $E_t = \emptyset$, r being an arbitrary start vertex. Then the blue rule is applied $n-1$ times by considering the cut $\text{Cut}(V_t)$ of all edges having one endpoint in t and the other outside t, yielding at each iteration a blue edge which is added to t.

- Kruskal's algorithm grows an empty forest f. At each of the $n-1$ iterations, the coloring rules are applied to the cheapest uncolored edge e. Namely, if e has both endpoints in f, it is colored red (it is the costliest edge in the cycle $\text{Cycle}(f, e)$), otherwise, it is colored blue (there is at least one cut in G containing no blue edge, where e is the cheapest uncolored edge) and added to t.

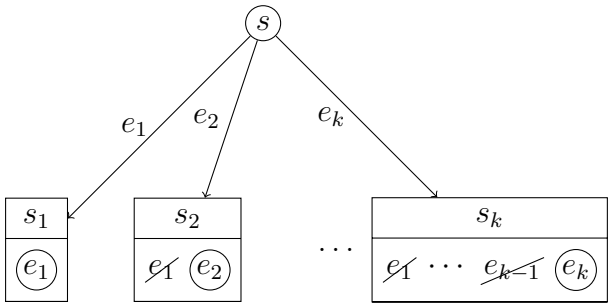


Figure 2.2: Non redundant generation of the children of a tree t given k edges $e_1, \ldots, e_k$. New excluded edges are crossed out, new included edges are circled

Enumeration scheme All generalizations presented below are implicit enumeration algorithms. They maintain at each iteration a set of partial solutions – trees or forests – each of which is augmented by adding edges, one at a time, yielding as many new partial solutions as candidate edges. To avoid redundancies, we must ensure that the partial solutions are disjoint. One way of satisfying this requirement is as follows:

- first define for each partial solution s and each edge e a status with respect to any descendant of s: included, excluded, or free;
- second, assume a partial solution s is to be augmented by adding one edge among a set of edges $e_1, \ldots, e_k$. An order is chosen on the candidates edges (e.g. the index) so as to exclude some edges in children partial solutions.

This simple scheme is illustrated in Figure 2.2.

The details discussed above are intentionally omitted in the description of the algorithms.

Generalization of Prim's algorithm

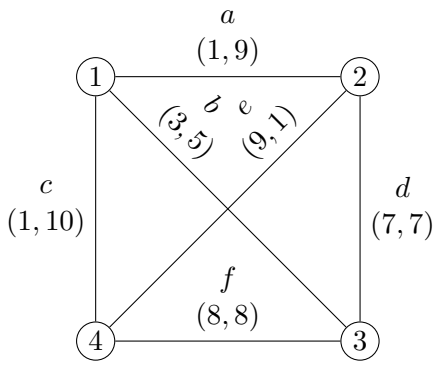
The algorithm of Corley (1985) This generalization is due to Corley (1985). The idea is to grow trees or connected forests by covering one more vertex at each iteration by selecting efficient edges in the cut $\mathrm{Cut}(V_f)$ formed by the sets of covered vertices V_f and non covered vertices $V \setminus V_f$. Algorithm 2.1 details this approach.

The justification follows directly from the first result of Theorem 2.2. However, since the conditions in this theorem are only necessary, non efficient spanning trees may be returned by the algorithm. Figure 2.3 shows an example application of this algorithm leading to some non efficient solutions.

An attempt to avoid the enumeration of forests leading to non efficient spanning trees Hamacher and Ruhe (1994) attempted to enhance Corley's algorithm by discarding at each of the $n-1$ iterations dominated forests, i.e. dominated trees in T_i are filtered

Algorithm 2.1: Generalization of Prim's algorithm by Corley (1985)

```
1 Let r be an arbitrary vertex of V
2 T_0 ← {(v_0, ∅)}; T_1 ← ··· ← T_{n−1} ← ∅
3 for i ← 1, ..., n − 1 do
4     for f = (V_f, E_f) ∈ T_{i−1} do
5         for e = (v, v') ∈ Cut(V_f)_eff such that v ∈ V_f do
6             T_i ← T_i ∪ {(V_f ∪ {v'}, E_f ∪ {e})}
7 return T_{n−1}
```



(a) Example graph

	Edges	Cost
T_1	$\{a\}$	$(1, 9)$
	$\{b\}$	$(3, 5)$
T_2	$\{a, b\}$	$(4, 14)$
	$\{a, c\}$	$(2, 19)$
	$\{a, e\}$	$(10, 10)$
	$\{b, d\}$	$(10, 12)$

	Edges	Cost
T_3	$\{a, b, c\}$	$(5, 24)$
	$\{a, b, e\}$	$(13, 15)$
	$\boldsymbol{\{a, b, f\}}$	$\boldsymbol{(12, 22)}$
	$\{b, d, c\}$	$(11, 22)$
	$\{b, d, e\}$	$(19, 11)$
	$\boldsymbol{\{b, d, f\}}$	$\boldsymbol{(18, 20)}$

(b) Algorithm trace (starts with vertex 1). Dominated trees shown in bold.

Figure 2.3: An example application of Algorithm 2.1 starting from $v_0 = 1$.

out after Step 6. The resulting algorithm was also presented in Zhou and Gen (1999) and Ehrgott (2005).

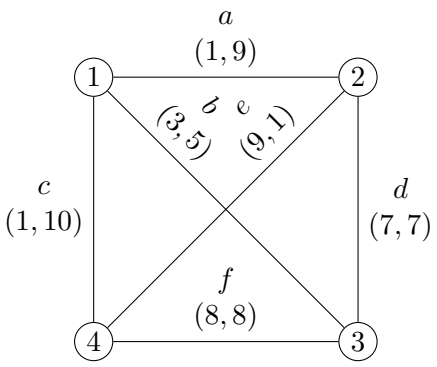
However, as remarked by Knowles and Corne (2002), some efficient spanning trees may be omitted. This also implies that some non efficient spanning trees may be returned by the algorithm and cannot be filtered. This is shown in Figure 2.4 where the algorithm is executed on the same instance as Algorithm 2.1.

Generalization of Kruskal's algorithm

The generalization comes from Serafini (1987). His algorithm relies on the following theorem.

Theorem 2.7 (Serafini, 1987)**.** *Let t be an efficient spanning tree. Then there exists an edge order on E compatible with the dominance relation $\precsim$ such that the greedy algorithm of Kruskal with this edge order returns t.*

Hence, the algorithm of Serafini (1987) tries all dominance-compatible edge orders. It is detailed in Algorithm 2.2.



(a) Example graph

	Edges	Cost
T_1	$\{a\}$	$(1, 9)$
	$\{b\}$	$(3, 5)$
T_2	$\{a, b\}$	$(4, 14)$
	$\{a, c\}$	$(2, 19)$
	$\{a, e\}$	$(10, 10)$
	~~$\{b, d\}$~~	~~$(10, 12)$~~

	Edges	Cost
T_3	$\{a, b, c\}$	$(5, 24)$
	$\{a, b, e\}$	$(13, 15)$
	$\mathbf{\{a, b, f\}}$	$\mathbf{(12, 22)}$
	~~$\{b, d, c\}$~~	~~$(11, 22)$~~
	~~$\{b, d, e\}$~~	~~$(19, 11)$~~
	~~$\mathbf{\{b, d, f\}}$~~	~~$\mathbf{(18, 20)}$~~

(b) Algorithm trace (starts with vertex 1). Dominated trees shown in bold. Sub-tree $\{b, d\}$ eliminated from T_2 although it would yield two efficient spanning trees.

Figure 2.4: An example application of the multi-objective Prim-like algorithm modified by Hamacher and Ruhe (1994)

Algorithm 2.2: A Kruskal-like multi-objective spanning tree algorithm by Serafini (1987)

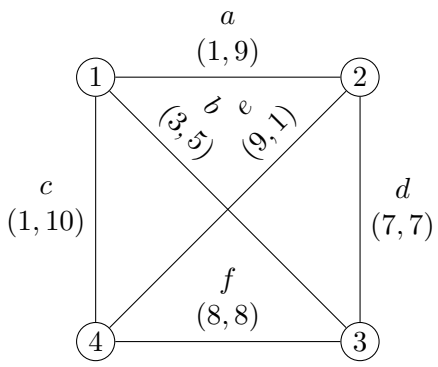
```
1 T_1 ← E_E                          – initialize a set of forest by selecting all efficient edges
2 T_2 ← ··· ← T_{n−1} ← ∅
3 for i ← 2, ..., n − 1 do
4     for f ∈ T_{i−1} do
5         C ← {e ∈ E \ f : f ∪ {e} does not contain a cycle} – define a set of candidate
          edges
6         for e ∈ C_eff do
7             T_i ← T_i ∪ {f ∪ {e}}
8 return T_{n−1}
```

An example application (on the same instance) is depicted in Figure 2.5.

Remarks

Perny and Spanjaard (2005) also considered generalizations of Prim's and Kruskal's algorithms in the case where any two subsets of the graph edges can be compared through a binary relation S. Their generalizations also maintain lists of forests and exclude S-dominated elements, just as with the dominance relation. They showed that their algorithms generate a superset of all minimal or maximal spanning trees if S satisfies the two following conditions:

- S is *quasi-transitive*, i.e. its asymmetric part P (for all $s, s' \subset E$, sPs' iff sSs' and



(a) Example graph

	Edges	Cost
T_1	$\{a\}$	$(1, 9)$
	$\{b\}$	$(3, 5)$
	$\{e\}$	$(9, 1)$
T_2	$\{a, b\}$	$(4, 14)$
	$\{a, c\}$	$(2, 19)$
	$\{a, e\}$	$(10, 10)$
	$\{b, d\}$	$(10, 12)$
	$\{b, e\}$	$(12, 6)$

	Edges	Cost
T_3	$\{a, b, c\}$	$(5, 24)$
	$\{a, b, e\}$	$(13, 15)$
	$\boldsymbol{\{a, b, f\}}$	$\boldsymbol{(12, 22)}$
	$\{b, d, c\}$	$(11, 22)$
	$\{b, d, e\}$	$(19, 11)$
	$\boldsymbol{\{b, d, f\}}$	$\boldsymbol{(18, 20)}$

(b) Algorithm trace. Dominated trees shown in bold.

Figure 2.5: An example application of Algorithm 2.2

not$(s'Ss)$) is transitive ;

- S satisfies the *independence axiom*, i.e. for all $s, s', s'' \subset E$ such that $s'' \cap (s \cup s') = \emptyset$, sSs' implies that $(s \cup s'')S(s' \cup s'')$.

The case of additive cost vectors compared through the dominance relation $\precsim$, which is also considered in Perny and Spanjaard (2005), obviously satisfies the two above conditions.

Comparison of the outputs of Algorithms 2.1 and 2.2 to the images of spanning trees compatible with the generalized coloring rules We saw that both Algorithms 2.1 and 2.2 and the generalized coloring rules produce dominated spanning trees. The application of the coloring rules on the same instance only makes it possible to color b, in blue. The spanning trees satisfying this coloring include the output of both algorithm, plus $\{b, c, e\}$ of cost $(13, 16)$ and $\{b, e, f\}$ of cost $(20, 14)$, which are dominated spanning trees. Hence, these algorithms produce in general smaller outputs than the coloring rules.

Worst case behavior of multi-objective Prim and Kruskal algorithms and coloring rules If no two edges dominate each other in the instance, multi-objective Prim and Kruskal algorithms (2.1 and 2.2) return all spanning trees of the instance. Similarly in this case, the coloring rules do not color any edge. Such instances may however contain non efficient spanning trees, as in the bi-objective instance of Figure 2.6.

This motivates the use of pruning rules in order to discard sub-trees which would lead to non efficient spanning trees. The failed attempt to enhance Corley's algorithm shows that pruning rules have to be carefully defined. The discussion in Section 1.2 provides some insight into this issue.

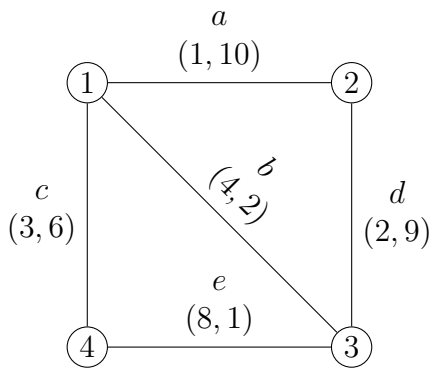


Figure 2.6: Example bi-objective instance where no two edges dominate each other but where not all spanning trees are efficient.
The spanning tree $t = \{a, d, e\}$ of cost $(11, 20)$ is dominated by the spanning tree $t' = \{a, b, c\}$ of cost $(8, 18)$.

2.4 Generic MOCO solving methods applied to the multi-objective Minimum Spanning Tree problem

Both the two-phase method and multi-objective branch and bound were instantiated for the MOST problem. The above two subsections survey the instantiations found in the literature.

2.4.1 Two-phase methods

We already presented bi-objective two-phase methods in Section 1.3.2. Both Ramos et al. (1998) and Steiner and Radzik (2003, 2008) tested two-phase methods on the Minimum Spanning Tree problem. The approach of Ramos et al. (1998) uses a branch and bound algorithm in phase two, which evaluates search nodes according to their ideal point in order to explore the search area defined from Y_{extr}, thus it can be viewed as an implicit enumeration algorithm initialized with the set of all extreme nondominated points. The approach of Steiner and Radzik (2008) relies on a ranking algorithm (namely, the algorithm of Gabow and Myers (1978)) to explore the same search area defined from all nondominated extreme points. Steiner and Radzik (2008) also provide experimental results showing the superiority of their approach to the one of Ramos et al. (1998) on random instances with uniform objective values.

2.4.2 A branch and bound algorithm

Sourd and Spanjaard (2008) proposed and implemented an enhanced branch and bound procedure which outperforms all earlier algorithms. We summarize the main enhancements

of their algorithm.

1. The instance is preprocessed by applying the generalized coloring rules presented in Section 2.3.1.

2. An initial set of solutions is obtained as the result of a local search procedure. Namely, starting from a set P of extreme efficient spanning trees for each extreme nondominated point, they compute the set of neighbors of all elements of P, that is the spanning trees which differ from those in P by a symmetric difference of their edge sets of 2. All efficient neighbors with respect to P are inserted into P while the dominated ones are removed from P. The procedure continues with the new spanning trees of P. While it is known since Ehrgott and Klamroth (1997) and Gorski et al. (2011) that nondominated points may be missed, the procedure often returns a reduced efficient set and provides a good initial search region. However, it yields many redundant computations and is rather time-consuming.

3. The set $Y(s)$ of all extensions of a given search node s is approximated by iteratively computing extreme nondominated points for s. This refines the bounding procedure of Ramos et al. (1998).

   This bounding of search nodes is also used in a *shaving* procedure. This procedure is run after the neighborhood search heuristic and aims at reducing the instance size. For each uncolored edge e, search nodes s_e and $s_{\bar{e}}$ are created where e is made respectively mandatory and forbidden. Then each search node s_e is evaluated. If it can be shown according to this evaluation that s_e cannot yield new nondominated points, then e is colored red. Otherwise $s_{\bar{e}}$ is considered and if it cannot yield new nondominated points, then e is colored blue.

4. As the infinite number of weight vectors $\lambda = (\lambda_1, \lambda_2)$ where $\lambda_1 \in [0, 1]$ and $\lambda_2 = 1 - \lambda_1$ induces only a polynomial number in m of edge orders (namely $\frac{m(m-1)}{2}$), they proposed to compute at the beginning of their procedure all these edge orders and store them in a priority queue using the first component λ_1 of associated weight vectors as key.

We remark that, to some extent, points 3 and 4 are specific to the bi-objective case.

Point 1 was adapted in Galand and Spanjaard (2012) for the computation of optimal spanning trees for the Ordered Weighted Average aggregation function.

2.5 Ranking spanning trees

The solutions to any 0-1 program can be ranked using the method of Murty (1968). The algorithm of Katoh et al. (1981) is a smart specialization of this generic method for the

Minimum Spanning Tree problem and its complexity is the best known when no assumption is made on the number of solutions to be computed. The complexity of obtaining the minimum cost solution together with some necessary computations to obtain next spanning trees is $O(\min(m \log n, n^2))$. Next solutions can be computed in only $O(m)$. The space requirement of the whole algorithm is $O(Km)$ and can be reduced to $O(K + m)$ where K is the number of computed solutions.

We assume in this section that the solutions are to be ranked according to an additive objective function $w : \mathcal{X} = 2^E \to \mathbb{R}$ (the weight of any subset of edges).

We first describe the partition scheme of Murty (1968) and the specialization proposed by Katoh et al. (1981). Then we present the procedures of Katoh et al. (1981).

2.5.1 Generic ranking scheme and specialization to the Minimum Spanning Tree problem

The method starts from an optimal solution $x^1 \in \{0,1\}^m$ and partitions the set of remaining solutions $X \setminus \{x^1\}$ into m subsets

$$\begin{aligned}
&\{x \in X : x_1 = 1 - x_1^1\} \\
&\{x \in X : x_1 = x_1^1, x_2 = 1 - x_2^1\} \\
&\qquad\qquad \vdots \\
&\{x \in X : x_1 = x_1^1, \ldots, x_{m-1} = x_{m-1}^1, x_m = 1 - x_m^1\}
\end{aligned}$$

Then a second best solution is found among those subsets which are subinstances of the initial instance. In general, when a k-best solution is found, the partition scheme is applied on the subinstance that brought it to obtain a $k+1$-best solution, which induces $O(m)$ new subinstances. Each of these subinstances will have to be solved. So the innovations in k-best algorithms lie in routines that efficiently optimize a subinstance starting from an optimal solution to the parent instance.

Consider the following measure of distance d between two solutions x and x': $d(x, x') = |\{i : x_i \neq x'_j\}|$. For general 0-1 programs, there is no constant upper bound on the smallest distance between a $k+1$-best solution and any of the computed k-best solutions. In the case of Minimum Spanning Tree problem however, we have the following result.

Proposition 2.8 (Deo, 1974; Katoh et al., 1981)**.** *The smallest distance between a $k+1$-best spanning tree and any of the computed k-best solutions is 2.*

The distance 2 corresponds to an edge exchange between two spanning trees.

Definition 2.9. For any spanning tree t, a t-exchange is any pair of edges (e, f) such that $f \in E \setminus t$ and $e \in \text{Cycle}(t, f)$.

The weight of (e, f) is $w(e, f) = w(f) - w(e)$.

The algorithm of Katoh et al. (1981) partitions instances following these principles:

1. any new spanning tree must differ from the previously computed spanning trees by at least one edge exchange,

2. sets of mandatory and forbidden edges, respectively $\text{in}(t)$ and $\text{out}(t)$, are maintained together with each previously computed spanning tree t in order not to obtain an already computed spanning tree.

Initially, a first spanning tree t_1 is computed and $\text{in}(t_1)$ and $\text{out}(t_1)$ are set to $\emptyset$. When a spanning tree t' is obtained by applying to a spanning tree t a *feasible* t-exchange (e, f), i.e. that satisfies $f \in E \setminus (t \cup \text{out}(t))$ and $e \in \text{Cycle}(t, f) \setminus \text{in}(t)$, we let $\text{in}(t') \leftarrow \text{in}(t) \cup \{f\}$ and $\text{out}(t') \leftarrow \text{out}(t)$, and we add f to $\text{out}(t)$. This ensures that t and t' cannot be obtained again by applying respectively to t' and t an edge exchange that satisfies the conditions represented by the in and out sets.

Assume that k best spanning trees $t_1, \ldots, t_k$ have been computed. A $k+1$-th best spanning tree is obtained by the best combination between a spanning tree t among $t_1, \ldots, t_k$ and a feasible t-exchange, i.e. as a solution to

$$\begin{cases} \min & w(t) + w(e, f) \\ \text{s.t.} & t \in \{t_1, \ldots, t_k\} \\ & f \in E \setminus (t \cup \text{out}(t)) \\ & e \in \text{Cycle}(t, f) \setminus \text{in}(t) \end{cases}$$

The correctness of the approach comes from Proposition 2.8 and the validity of the partition scheme through in and out sets.

So to rank spanning trees according to Katoh et al. (1981), apart from the update of in and out sets, it is required to:

1. compute a minimum weight spanning tree t_1 and a minimum weight t_1-exchange (e, f) for each $f \in E \setminus t_1$,

2. for any spanning tree t' obtained from a previously computed spanning tree t, compute a minimum weight feasible t'-exchange for each non tree edge of t',

3. retrieve the current best spanning tree – edge exchange combination.

We now describe and discuss the proposals of Katoh et al. (1981) for these needs.

2.5.2 Procedures of the ranking algorithm of Katoh et al. (1981)

Computation of t_1 and required t_1-exchanges To obtain t_1, a first minimum cost spanning tree together with its edge exchanges, two procedures are presented in Katoh et

al. (1981) whose complexities are $O(n^2)$ and $O(m \log n)$. The first procedure uses Prim's algorithm to obtain t_1. At each iteration, it also maintains at no extra cost (w.r.t. the big O notation) for each non tree edge f, the maximum weight edge in Cycle(t_1, f). Thanks to these computations, the required edge exchanges are then immediately obtained.

The second procedure uses Kruskal's algorithm to obtain t_1 and then the spanning tree verification algorithm found in Tarjan (1979) to generate the associated edge exchanges. The complexity of this approach is $O(m \log n)$.

Computation of t'-exchanges associated to a spanning tree obtained from t Some but not all t-exchanges will remain feasible for t'. Katoh et al. (1981) propose an algorithm which computes the missing t'-exchanges in $O(m)$. This is basically done by performing two depth first searches in $t \setminus \{f\}$: one from each endpoint of f. We omit the detail since there is not much to discuss here.

Retrieval of the current best spanning tree – edge exchange combination In order to efficiently obtain the current best spanning tree – edge exchange combination, all previously computed spanning trees are stored together with their minimum weight feasible edge exchanges in a priority queue. Namely, for any spanning tree t among the first k-best trees that admits a minimum weight feasible t-exchange (e, f), the pair $(t, (e, f))$ is stored with the value $w(t) + w(e, f)$ as key. For each $t \in \{t_1, \ldots, t_k\}$, the minimum weight feasible t-exchanges associated to each non tree edge are also maintained in a priority queue together with t. Both types of priority queues have to support insertions and deletions. Any binary heap can be used and the time for each insertion and deletion is bounded above by $O(m)$.

Conclusions

This chapter addressed the multi-objective version of a classical problem in operations research, which is the Minimum Spanning Tree problem. We discussed the complexity of generating the efficient set and deciding whether a solution is efficient, both for the multi-objective Minimum Spanning Tree problem (MOST). We considered the generalization to the multi-objective case of well-known optimality conditions for the single objective version of the problem. They yield preprocessing rules which can possibly simplify an instance of the MOST problem as well as multi-objective versions of two classical algorithms for the Minimum Spanning Tree problem. However, the generalized algorithms are not interesting in practice since they may generate many non efficient solutions. This raises the interest in other approaches to generate the nondominated set for the MOST problem. Therefore, we surveyed earlier instantiations on the MOST problem of generic MOCO solving methods.

Chapter 3

Determining the search region in multi-objective optimization

Given a set N of feasible points of a multi-objective optimization (MOO) problem, the *search region* corresponds to the part of the objective space containing all the points that are not dominated by any point of N, i.e. the part of the objective space which may contain further nondominated points. In this chapter, we consider a representation of the search region by a set of tight *local upper bounds* (in the minimization case) that can be derived from the points of N. Local upper bounds play an important role in methods for generating or approximating the nondominated set of an MOO problem, yet few works in the field of MOO address their efficient incremental determination. We relate this issue to the state of the art in computational geometry and provide several equivalent definitions of local upper bounds that are meaningful in MOO. We discuss the complexity of this representation in arbitrary dimension, which yields an improved upper bound on the number of solver calls in epsilon-constraint-like methods to generate the nondominated set of a discrete MOO problem. We analyze and enhance a first incremental approach which operates by eliminating redundancies among local upper bounds. We also study some properties of local upper bounds, especially concerning the issue of redundant local upper bounds, that give rise to a new incremental approach which avoids such redundancies. Finally, the complexities of the incremental approaches are compared on the theoretical and empirical points of view.

This chapter is adapted from Klamroth et al. (2014) which has been submitted for publication.

Contents

3.1 Background and motivations 52
3.1.1 Multi-objective optimization setting 52
3.1.2 The search region 53
3.1.3 Explicit representation of the search region by local upper bounds . . 53
3.1.4 Related concepts 56
3.1.5 Application for the solution of MOO problems 57
A generic method based on the solution of budget constrained programs • MOBB and two-phase methods
3.2 Existence and construction of upper bound sets 59
3.3 Properties of local upper bounds and their efficient computation 63
3.3.1 Introductory examples and geometric interpretation 63
3.3.2 Theoretical properties of local upper bounds and a new incremental approach 66
3.4 Complexity and computational experiments 70
3.4.1 Tight upper bound on the number of local upper bounds 71
3.4.2 Worst-case complexities of the algorithms 71
Common steps of both algorithms • Remaining steps
3.4.3 Experimental comparison of the algorithms 73
Experimental setup • Observations on the SA instances • Comparison of the algorithms

Introduction

Most solution approaches in multi-objective optimization (MOO) aimed at outputting a set of "good" solutions iteratively generate candidate solutions. Generally, a pool of solutions is maintained and updated when new solutions arrive. The pool provides information which is used to decide whether a new solution should be inserted and whether old solutions should be removed. It can also be used to guide the search process within the objective space. In particular, from the images in the objective space of the pool solutions, we can define the part of the objective space containing all points that none of these images dominate, which we refer to as the *search region*.

The concept itself is well known in the field. Especially in the two dimensional or bi-objective case, it is a key tool of the two-phase and branch and bound methods. In the two-phase method (see Ulungu and Teghem, 1995), adjacent extreme nondominated points computed in the first phase define triangles which delimit zones where all other nondominated points lie. The so-called *local nadir* points corresponding to the right angles of these triangles act as *local upper bounds* that together define the search region, assuming that the objectives are to be minimized. This upper bounding part is also one of the foundations of multi-objective branch and bound (see Sourd and Spanjaard (2008)). Actually, the representation of the search region through local upper bounds makes it possible to test the existence of the intersection between the search region and a convex lower bound on the feasible points associated to a search node and to decide whether to fathom the search node or not.

Given a discrete set of points $Q \subseteq \mathbb{R}^p$, Kaplan et al. (2008) consider *maximal empty orthants* with respect to Q, which contain no point of Q and are maximal for this property under inclusion. Assuming that the points of Q are feasible points of an MOO problem, the union of all maximal empty orthants corresponds to the search region associated to Q and their apexes correspond to local upper bounds in the context of MOO. Kaplan et al. (2008) give an algorithm for the generation of all maximal empty orthants, and hence for the computation of all local upper bounds of the search region. However it requires that the input points are given in a nondecreasing order of some component and in this sense does not directly imply an incremental approach.

Przybylski et al. (2010b) considered local upper bounds in arbitrary dimension for a generalization of the two-phase method to problems with arbitrarily many objectives. They propose an online algorithm to carry out the update of the local upper bounds as soon as new feasible points are discovered, but they do not consider the complexity of this operation.

Several solution methods to generate the nondominated set iteratively solve linear programs parametrized by local upper bounds (see Sylva and Crema (2007)), possibly including redundancies (see Lokman and Köksalan (2013) and Kirlik and Sayın (2014)). In Sylva and Crema (2007) each local upper bound is determined by solving an integer linear program.

Also, Dächert and Klamroth (2014) proposed to compute *boxes* for three dimensional MOO problems that are defined by a common lower bound and several upper bounds and decompose the search region. They developed an efficient incremental algorithm, that avoids redundancies, to update the decomposition each time a nondominated point is found, through e.g. the optimization of a pseudo-distance function parametrized by the defining points of the box. In particular, they showed that, in the three dimensional case, the search region can be described by $2n + 1$ boxes if the number of known feasible points is n.

The chapter is organized as follows. Section 3.1 sets some notations, formally defines the concepts of *search region* and *upper bound set*, then motivates their use in MOO. Section 3.2 shows the existence and uniqueness of local upper bounds through a first algorithm for which we discuss some enhancements. Section 3.3 investigates some properties of the elements of upper bound sets that yield another approach to compute an upper bound set. Section 3.4 is devoted to the complexity aspects related to the representation of the search region by a set of local upper bounds and the comparison of the two approaches from both theoretical and empirical points of view. The last section provides conclusions and perspectives.

3.1 Background and motivations

3.1.1 Multi-objective optimization setting

We consider multi-objective optimization (MOO) problems as in Chapter 1

$$\begin{cases} \min & c(x) = (c_1(x), \dots, c_p(x)) \\ \text{s.t.} & x \in X \end{cases} \tag{3.1}$$

with feasible set $X \neq \emptyset$ and with $p \geq 2$ objective functions $c_j : X \to \mathbb{R}$, $j = 1, \dots, p,$. For all $j \in \{1, \dots, p\}$ and $x \in X$, we assume, for any instance of an MOO problem, that $m < c_j(x) < M$ for some $m, M \in \mathbb{R}$, or that such values m and M exist that bound the area of interest for the decision maker. We will refer to $Z = (m, M)^p$ as the p-dimensional *search interval*, a set that contains all feasible or at least all relevant points. We denote by $\hat{Z} = [m, M]^p$ the closure of Z.

We define some general notations. We denote by $\boldsymbol{M}$ the p-dimensional vector $(M, \dots, M)$ and analogously the p-dimensional vector $\boldsymbol{m} = (m, \dots, m)$ and the p-dimensional all-ones vector $\mathbf{1}$. For any $z \in \mathbb{R}^p$, we let z_{-j} be the $(p-1)$-dimensional vector of all components of z excluding component j, for a given $j \in \{1, \dots, p\}$. Finally, for any $z, a \in \mathbb{R}^p$ and any $j \in \{1, \dots, p\}$, (z_j, a_{-j}) denotes the vector $(a_1, \dots, a_{j-1}, z_j, a_{j+1}, \dots, a_p)$. Such a vector will be referred to as the jth projection of vector z on vector a.

3.1.2 The search region

In the following definition, we formalize the concept of *search region* which we presented in the introduction.

Definition 3.1. Let N be a finite and stable set of feasible points. The *search region* for $Y_{\text{nd}} \setminus N$, denoted by $S(N)$, contains all the points in Z that could be nondominated given N, or alternatively, excludes all the points in Z that are dominated by at least one point in N, that is:

$$\begin{aligned} S(N) &= \{z \in Z : \forall \bar{z} \in N,\, \bar{z} \not\leqq z\} \\ &= Z \setminus \{z \in Z : \exists \bar{z} \in N \text{ with } \bar{z} \leqq z\} \end{aligned} \tag{3.2}$$

Note that the search region $S(N)$ excludes the points in N, since they are already known, that is, $S(N) \cap N = \emptyset$.

In some cases, N is a subset of Y_{nd} obtained by some scalarizing function. More generally, N may contain any feasible point, no matter how it is obtained, e.g. by any heuristic procedure. It could even be any stable set of not necessarily feasible points from Z, provided none of its points dominate any point of the nondominated set Y_{nd}.

Regarding the stability condition on N, it can be easily seen that the set $S(N)$ is not affected if a point dominated by another point of N is added. In other words:

Remark 3.2. For any set of points Q, we have $S(Q) = S(Q_{\text{nd}})$, i.e., both sets induce the same search region.

Consequently, the assumption that the set N is stable can be made without loss of generality.

3.1.3 Explicit representation of the search region by local upper bounds

Our purpose is to find an explicit and concise characterization of $S(N)$ using a finite set $U(N)$ of minimal *local upper bounds*, which could also be referred to as *local nadir points* or *maximal points* (for the dominance relation). We will refer to $U(N)$ as an *upper bound set* for the search region $S(N)$ in the following.

Every local upper bound $u \in U(N)$ defines a *search zone* $C(u) \subset Z$ as

$$C(u) = \{z \in Z : z < u\},$$

and the search region $S(N)$ is covered by the union of these search zones. In order to possibly include any point of Z in a given search zone $C(u)$, the possible values for u should include the boundary of Z. So in general $U(N)$ is a subset of $\hat{Z}$, the closure of the search interval Z.

In the following, we give three alternative definitions for upper bound sets and show their equivalence.

Definition 3.3. Let $N \subset Z$ be a finite and stable set of points. A set $U(N) \subset \hat{Z}$ is called an *upper bound set with respect to* N if and only if

(1) $S(N) = \bigcup_{u \in U(N)} C(u)$ and

(2) $\forall u^1, u^2 \in U(N),\ C(u^1) \not\subset C(u^2)$.

While condition (1) in Definition 3.3 guarantees that the search region $S(N)$ is *exactly* represented by the search zones induced by $U(N)$, condition (2) ensures minimality of the set $U(N)$ in the sense that no redundant search zones are contained in the representation. Observe that this definition can be seen as a natural extension of the concept of *upper bound* in the 1-dimensional case. If $p = 1$, a stable set may either be empty ($N = \emptyset$), or it may consist of exactly one point ($N = \{\bar{z}\}$). The corresponding search region is then uniquely represented by one point, namely $\bar{u} = \boldsymbol{M}$ in the first case and $\bar{u} = \bar{z}$ in the latter case.

As an example, we describe the situation in the 2-dimensional case.

Example 1. Let $N = \{(z_1^1, z_2^1), \ldots, (z_1^n, z_2^n)\}$ be a stable set of 2-dimensional points (with $n \geq 1$). In the bi-objective case, the points in any stable set N can be ordered such that the objective values are strictly increasing in the first objective and strictly decreasing in the second objective. Hence we can assume that $z_1^1 < \cdots < z_1^n$ and $z_2^1 > \cdots > z_2^n$. The search region consists of the union of search zones defined by pairs of consecutive points in N. Thus the upper bound set associated to N is

$$U(N) = \{(z_1^1, M), (z_1^2, z_2^1), (z_1^3, z_2^2), \ldots, (z_1^n, z_2^{n-1}), (M, z_2^n)\}$$

We illustrate this example in Figure 3.1.

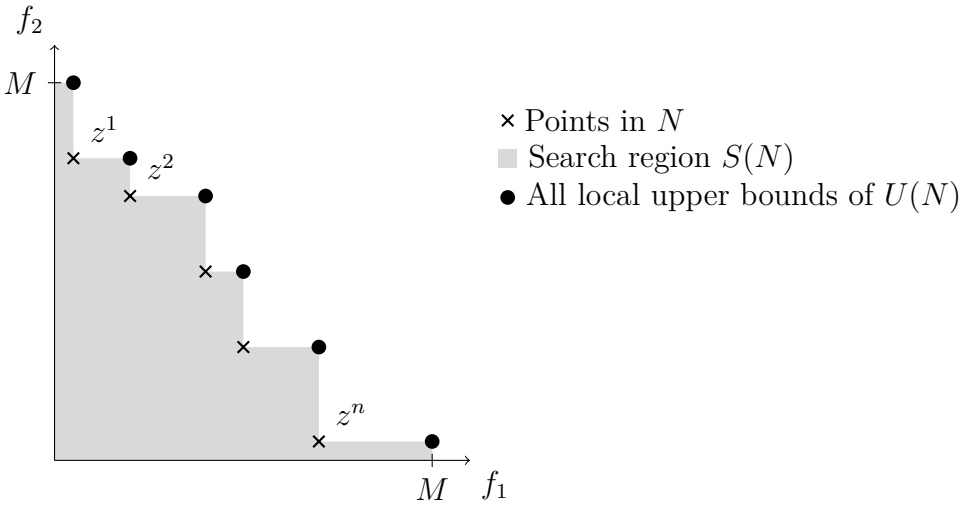


Figure 3.1: Illustration of the concepts of *search region* and *local upper bound* ($p = 2$)

The conditions of Definition 3.3 can immediately be reformulated in terms of pairwise comparisons between points in $S(N)$ and $U(N)$.

Proposition 3.4. *Let $N \subset Z$ be a finite and stable set of points. Then $U(N) \subset \hat{Z}$ is an upper bound set with respect to N if and only if*

(1a') $\forall z \in S(N)\, \exists u \in U(N) : z < u$,

(1b') $\forall z \in Z \setminus S(N)\, \forall u \in U(N) : z \not< u$, *and*

(2') $\forall u^1, u^2 \in U(N) : u^1 \not\leqq u^2$.

Proof. Conditions (1a') and (1b') together are equivalent to condition (1) in Definition 3.3, and condition (2') is equivalent to condition (2) in Definition 3.3. □

In the case where the search interval is restricted to integer-valued vectors, i.e. $Z \subset \mathbb{Z}^p$, conditions (1a') and (1b') of Proposition 3.4 can be further specified since for all $z, z' \in \mathbb{Z}^p$ such that $z < z'$, we have $z \leqq z' - \mathbf{1}$. We briefly restate them in the following remark.

Remark 3.5. Assume $Z \subset \mathbb{Z}^p$ and $\boldsymbol{M} \in \mathbb{Z}^p$. Under the same hypothesis of Proposition 3.4, we have for the upper bound set $U(N) \subset \hat{Z}$:

(1a") $\forall z \in S(N)\, \exists u \in U(N) : z \leqq u - \mathbf{1}$ and

(1b") $\forall z \in Z \setminus S(N)\, \forall u \in U(N) : z \not\leqq u - \mathbf{1}$

This is particularly useful in the context of the two-phase and branch and bound algorithms which we discuss at the end of this section.

A yet alternative characterization of local upper bounds that will turn out useful for their efficient determination is given in Proposition 3.6. In particular, local upper bounds are exactly those points that (i) are not strictly dominated by any of the points in N, and (ii) are maximal with this property.

Proposition 3.6. *Let $N \subset Z$ be a finite and stable set of points. Then $U(N) \subset \hat{Z}$ is an upper bound set with respect to N if and only if $U(N)$ consists of all points $u \in \hat{Z}$ that satisfy the following two conditions:*

(i) no point of N strictly dominates u and

(ii) for any $\bar{u} \in \hat{Z}$ such that $\bar{u} \geq u$, there exists $\bar{z} \in N$ such that $\bar{z} < \bar{u}$, i.e., u is a maximal point with property (i).

Proof. Let $U(N)$ denote the upper bound set with respect to N, and let $U'(N)$ denote the set of all points satisfying (i) and (ii) above.
Claim 1: $U(N) \subset U'(N)$. Let $u \in U(N)$. We show that u satisfies (i) and (ii).

(i) Assume that there exists a point $\bar{z} \in N$ such that $\bar{z} < u$. Then by condition (1) of Definition 3.3, $\bar{z} \in C(u) \subset S(N)$, which contradicts $S(N) \cap N = \emptyset$.

(ii) Let $\bar{u} \geq u$ and hence $C(u) \subset C(\bar{u})$. Since $u \in U(N)$, by condition (2) of Definition 3.3, we get $\bar{u} \notin U(N)$. Thus, there exists a point $z' \in C(\bar{u})$ such that $z' \in Z \setminus S(N)$. Since $z' \in Z \setminus S(N)$, there exists a point $\bar{z} \in N$ with $\bar{z} \leqq z'$ and hence $\bar{z} < \bar{u}$.

Claim 2: $U'(N) = U(N)$, i.e., we show that $U'(N)$ satisfies (1) and (2).

First observe that for any $u' \in U'(N)$, we have $C(u') \subset S(N)$. Indeed, if there exists $z' \in C(u') \setminus S(N)$, then there exists $z \in N$ such that $z \leqq z'$. Since $z' < u'$, we get $z < u'$ contradicting condition (i).

(1) Let $u' \in U'(N)$. Then we have $C(u') \subset S(N)$ and hence $\cup_{u' \in U'(N)} C(u') \subset S(N)$. From Claim 1 above, we have $S(N) = \cup_{u \in U(N)} C(u) \subset \cup_{u' \in U'(N)} C(u')$, and thus $\cup_{u' \in U'(N)} C(u') = S(N)$, which proves (1).

(2) Now let $u^1, u^2 \in U'(N)$. Then we have $C(u^1) \subset S(N)$ and $C(u^2) \subset S(N)$. If $C(u^1) \subset C(u^2)$, then (ii) would be violated. This proves (2).

□

3.1.4 Related concepts

In computational geometry, Kaplan et al. (2008) define *maximal empty orthants* with respect to a discrete set of points $Q \subseteq \mathbb{R}^p$ as partially bounded hyperrectangles of the form $\prod_{j=1}^{p}(-\infty, a_j) \subseteq \mathbb{R}^p$, for some $a \in \mathbb{R}^p$ (apex), which contain no point of Q and are maximal for this property under inclusion. It is clear that such points a satisfy the conditions of Proposition 3.6 and are therefore local upper bounds for the search region $S(Q)$.

For their generalization of the two-phase method to MOO problems with more than two objectives, Przybylski et al. (2010b) are interested in characterizing the part of the objective space where remaining nondominated points have to be searched after phase one. To this end, they defined a concept similar to our search region, the *search area*. The search area $S'(N)$ is defined as the closure of the complement set of $\{z \in \mathbb{R}^p : \exists z' \in N, z' \leqq z\}$, i.e.,

$$S'(N) = \mathrm{cl}\left(\{z \in \mathbb{R}^p : \exists z' \in N, z' \leqq z\}^C\right).$$

N is defined in Przybylski et al. (2010b) as an *upper bound set for the nondominated set* in the sense of Ehrgott and Gandibleux (2007), and can also be any stable set of feasible points. Note that we omit from their definition a lower bounding part, which is not relevant for our purpose.

In fact, the search area $S'(N)$ corresponds to the closure of the search region $S(N)$ defined according to equation (3.2) in Definition 3.3. This difference implies that the search

area $S'(N)$ includes N and even points of the objective space that are weakly dominated by some points of N.

Przybylski et al. (2010b) and Dächert and Klamroth (2014) also describe the search area by a set of *corner points* or *upper bounds* which are the same as the local upper bounds we consider in this chapter. The former rely on a definition for these points which corresponds to Proposition 3.6.

3.1.5 Application for the solution of MOO problems

The concepts and properties developed in Sections 3.1.2 and 3.1.3 apply to MOO in general. For continuous and mixed discrete-continuous problems, they are useful in approaches aimed at generating discrete representations of the nondominated set. In the case of discrete problems, such as multi-objective combinatorial optimization (MOCO) problems, they play an important role in the generation of the nondominated set as well. We focus in this section on the latter issue and mention two widely applied methods to show how the computation of local upper bounds can be integrated into an overall solution strategy.

A generic method based on the solution of budget constrained programs

The representation of the search region as a set of search zones makes it possible to derive a simple algorithm to enumerate all nondominated points of a MOCO problem. This can be done by iteratively exploring the search zones that define the search region and updating the search region whenever new points are found. The exploration of a search zone $C(u)$ has to determine whether $C(u)$ contains feasible points, and if so output one such point. In order to limit the number of search zones that are considered, the exploration routine should return only nondominated points. Such an exploration can be achieved by solving, for example, the following mathematical program associated to a search zone $C(u)$:

$$P(u) : \min\{g(c(x)) : x \in X, c(x) < u\}$$

where g is any strongly increasing aggregation function of the c_j's (e.g. $g : z \mapsto \sum_{j=1}^{p} z_j$). Note that the strict dominance in the definition of $P(u)$ can be translated into non-strict inequalities, e.g. in the case of MOCO by slightly decreasing u since then X is a discrete set, or, when possible, by taking advantage of Remark 3.5.

The generic method is presented in Algorithm 3.1. From property (2) of Definition 3.3, there is no redundant constrained program among the programs associated to the search zones of the current search region.

Note that each local upper bound that is considered at Step 3 will either lead to a nondominated point or be part of the final upper bound set $U(Y_{\text{nd}})$ (if the associated $P(u)$ has no feasible solution). Therefore the number of calls to the exploration routine is

Algorithm 3.1: Generic method to generate all nondominated points of a MOCO problem based on the definition of search zones

input : X, c, $\boldsymbol{M}$
output : Y_{nd}

$N \leftarrow \emptyset$; $U(N) \leftarrow \{\boldsymbol{M}\}$
while $U(N) \neq \emptyset$ **do**
  Select $u \in U(N)$
  if $P(u)$ *is feasible* **then**
    Let $\bar{z}$ be an optimal point of $P(u)$
    $N \leftarrow N \cup \{\bar{z}\}$
    Update $U(N)$
  else
    $U(N) \leftarrow U(N) \setminus \{u\}$
return $N = Y_{\text{nd}}$

exactly $|U(Y_{\text{nd}})| + |Y_{\text{nd}}|$. This, together with the tight upper bound on $|U(Y_{\text{nd}})|$ provided in Section 3.4.1, amounts to $O(|Y_{\text{nd}}|^{\lfloor \frac{p}{2} \rfloor})$ solver calls for $p \geq 2$.

Several works (Kirlik and Sayın (2014), Laumanns et al. (2006), Lokman and Köksalan (2013), Özlen and Azizoglu (2009), and Sylva and Crema (2008)) develop similar methods based on the solution of budget constrained programs, sometimes considered as generalizations of the ε-constraint method to the case $p \geq 3$, but they do not provide a non-trivial upper bound on the number of solver calls. Dächert and Klamroth (2014) show, however, using a similar approach that, in the particular case $p = 3$, the number of calls to the exploration routine is upper bounded by $3|Y_{\text{nd}}| - 2$.

MOBB and two-phase methods

In multi-objective branch and bound (MOBB), we usually perform a bounding step at each node of a search tree. Assume we consider the current node whose set of feasible solutions is $X' \subset X$. In general, computing either X' or $Y' = c(X')$ would be expensive. However, given a set of, say m weight vectors $\lambda^1, \ldots, \lambda^m$ of $\mathbb{R}^p$ such that for any $i = 1, \ldots, m$, $\lambda^i \geq \mathbf{0}$, we may approximate Y' by computing $\alpha_i = \min\{\sum_{j=1}^{p} \lambda_j^i z_j : z \in Y'\}$, especially if the single objective version of the underlying problem is solvable in polynomial time. Denoting by Q the set $\{z \in Z : \sum_{j=1}^{p} \lambda_j^i z_j \geq \alpha_i,\ i = 1, \ldots, m\}$, we have $Y' \subset Q$. Therefore, if $Q \cap S(N) = \emptyset$ then the current node can be pruned since it cannot yield any new nondominated point.

Consider also the two-phase method, and especially the version where a ranking algorithm is used to obtain, in phase two, nondominated non-extreme points. A set of weight vectors $\lambda^1, \ldots, \lambda^m$ that satisfy the same conditions as above is obtained in phase one. At some time during phase two, we are given m values $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$ such that for

each $i = 1, \ldots, m$, all feasible points whose weighted sum value according to the weight vector λ^i is less than or equal to α_i have been computed. Considering N as the set of all these feasible points, excluding the dominated ones, we can test whether the set $Q = \{z \in Z : \sum_{j=1}^{p} \lambda_j^i z_j \geq \alpha_i,\ i = 1, \ldots, m\}$ intersects the search region $S(N)$.

Now we explain how local upper bounds help to determine whether such a polytope Q intersects the search region. For any $\lambda \in \mathbb{R}^p$ such that $\lambda \geq \mathbf{0}$ and for any $z, z' \in \mathbb{R}^p$ such that $z < z'$, we have $\sum_{j=1}^{p} \lambda_j^i z_j < \sum_{j=1}^{p} \lambda_j^i z'_j$. Therefore, together with conditions (1a') and (1b') of Proposition 3.4, we obtain:

$$Q \cap S(N) = \emptyset \text{ if } \forall u \in U(N),\ \exists i \in \{1, \ldots, m\} \text{ such that } \sum_{j=1}^{p} \lambda_j^i u_j \leq \alpha_i \qquad \text{(Rule "}\mathbb{R}\text{")}$$

If the feasible points are integral, that is we restrict ourselves to integer vectors in both $S(N)$ and Q, the above condition for $Q \cap S(N) = \emptyset$ can be strengthened using Remark 3.5. In this case, we rely on the following implication: if z and z' are two vectors such that $z \leqq z'$, then $\sum_{j=1}^{p} \lambda_j^i z_j \leq \sum_{j=1}^{p} \lambda_j^i z'_j$. Then in this case, we have:

$$Q \cap S(N) = \emptyset \text{ if } \forall u \in U(N),\ \exists i \in \{1, \ldots, m\} \text{ such that } \sum_{j=1}^{p} \lambda_j^i u_j < \alpha_i - \sum_{j=1}^{p} \lambda_j^i \quad \text{(Rule "}\mathbb{Z}\text{")}$$

These rules are used by Sourd and Spanjaard (2008) in the context of MOBB to find all nondominated points of the bi-objective minimum spanning tree problem. Przybylski et al. (2008) also use them in a two-phase method based on the use of a ranking algorithm to find all nondominated points of the bi-objective assignment problem. Przybylski et al. (2010b) consider their application again in a two-phase method not limited to the bi-objective case. The rules they propose for the general multi-objective case are related to their definition of the search area we presented in Section 3.1.4, which implies that the rule for the integral case is a little weaker than Rule "$\mathbb{Z}$".

3.2 Existence and construction of upper bound sets

The initial search region consists of the whole search interval Z. Therefore, it can be described by the following upper bound set:

$$U(\emptyset) = \{\boldsymbol{M}\}.$$

Actually, this defines the unique search zone $C(\boldsymbol{M}) = Z$, which is consistent with Definition 3.3.

Starting with this in the case $N = \emptyset$, a simple incremental algorithm can be formulated

that iteratively introduces points to the set N and updates the upper bound set accordingly. It was first proposed by Przybylski et al. (2010b) with a slight difference in the filtering step to which we shall return later. Given a finite and stable set $N \subset Z$, a corresponding upper bound set $U(N)$, and a point $\bar{z} \in Z$ that is nondominated with respect to N, Algorithm 3.2 describes the updating procedure to obtain the upper bound set $U(N \cup \{\bar{z}\})$.

Algorithm 3.2: Update procedure of an upper bound set based on redundancy elimination

input : $U(N)$, $\bar{z}$ – *Set of local upper bounds and new point*
output : $U(N \cup \{\bar{z}\})$

1 $A \leftarrow \{u \in U(N) : \bar{z} < u\}$ – *Search zones that contain* $\bar{z}$
2 $B \leftarrow \{u \in U(N) \setminus A : \bar{z} \leq u\}$ – *Search zones whose boundary contains* $\bar{z}$
3 $P \leftarrow \emptyset$
4 **for** $u \in A$ **do**
5 **for** $j \in \{1, \ldots, p\}$ **do**
6 $P \leftarrow P \cup \{(\bar{z}_j, u_{-j})\}$ – *Generate all projections of* $\bar{z}$ *on the local upper bounds of* A
7 $P \leftarrow \{(\bar{z}_j, u_{-j}) \in P : (\bar{z}_j, u_{-j}) \not\leq u', \forall u' \in P \cup B\}$ – *Filter out all redundant points of* P
8 $U(N \cup \{\bar{z}\}) \leftarrow (U(N) \setminus A) \cup P$

Basically, Algorithm 3.2 updates each search zone $C(u)$ in which the new point $\bar{z}$ lies by removing from $C(u)$ the part of Z which is dominated by $\bar{z}$ (including $\bar{z}$). This is achieved by replacing $C(u)$ by p subzones as done in Step 6 of Algorithm 3.2. Some of these newly generated subzones may be redundant, and are thus removed in Step 7. More formally, we state the following result, which justifies Algorithm 3.2.

Proposition 3.7. *Let $N \subset Z$ be a non-empty finite and stable set of points. Applying Algorithm 3.2 iteratively on the points of N, starting with an initial upper bound set $U(\emptyset) = \{\boldsymbol{M}\}$, returns the correct upper bound set $U(N)$.*

Proof. We show that Algorithm 3.2 correctly computes the set $U(N \cup \{\bar{z}\})$, given any finite and stable set $N \subset Z$ of points, the correct upper bound set $U(N)$, and a new point $\bar{z} \in Z$ that is nondominated with respect to N. The result then follows by induction.

Considering the new point $\bar{z}$, the search region $S(N \cup \{\bar{z}\})$ must be updated from $S(N)$ by removing all points in Z such that $\bar{z} \leqq z$.

In Step 1 of Algorithm 3.2 the search zones $C(u)$, $u \in A$, containing $\bar{z}$ are identified. All other search zones $C(u)$, $u \in U(N) \setminus A$, are not affected by the new point $\bar{z}$ and thus need not be modified.

Thus, we just need to remove the set of points $\{z \in S(N) : \bar{z} \leqq z\}$ from the search zones $C(u)$, $u \in A$, to ensure that condition (1) of Definition 3.3 is satisfied. Steps 4-6 are

justified by the fact that for any $u \in A$ we have

$$C(u) \setminus \{z \in S(N) : \bar{z} \leqq z\} = \bigcup_{j=1}^{p} C(\bar{z}_j, u_{-j}).$$

Among the candidate local upper bounds of P, as defined after all iterations of Step 6, there may be some *redundant* local upper bounds in the sense that they induce search zones that are included in a search zone associated to some (candidate) local upper bounds of $P \cup (U(N) \setminus A)$. Let $(\bar{z}_j, u_{-j}) \in P$, with $u \in A$, be a redundant local upper bound, i.e. there exists $u' \in P \cup (U(N) \setminus A)$ such that $\bar{z} \leq (\bar{z}_j, u_{-j}) \leq u'$. If $\bar{z} < u'$, then P contains the candidate local upper bound $(\bar{z}_j, u'_{-j})$, otherwise we have $u' \in B$. Therefore, Step 7 correctly filters the set P, which leads to satisfying condition (2) of Definition 3.3. □

In Przybylski et al. (2010b), the filtering step is formulated with respect to the set $U(N)$, i.e.

$$P \leftarrow \{(\bar{z}_j, u_{-j}) \in P : (\bar{z}_j, u_{-j}) \not\leq (u'), \forall u' \in U(N)\}.$$

This is correct, but involves unnecessary dominance tests compared to Algorithm 3.2, since one only needs to filter with respect to $P \cup B$ instead of $U(N)$.

It is even possible to further refine the filtering step of Algorithm 3.2. To this end, we prove the following proposition.

Proposition 3.8. *Let $(\bar{z}_j, u_{-j})$ be a candidate local upper bound in P. Then:*

(1) $(\bar{z}_j, u_{-j}) \leq (\bar{z}_k, u'_{-k})$ for some $(\bar{z}_k, u'_{-k}) \in P$ with $k \neq j$ implies $(\bar{z}_j, u_{-j}) \leq (\bar{z}_j, u'_{-j})$;

(2) $(\bar{z}_j, u_{-j}) \leq u'$ for some $u' \in B$ implies $\bar{z}_j = u'_j$ and $\bar{z}_{-j} < u'_{-j}$.

Proof.

(1) Since $\bar{z} < u'$, we have $(\bar{z}_k, u'_{-k}) \leq u'$, which, together with $(\bar{z}_j, u_{-j}) \leq (\bar{z}_k, u'_{-k})$, leads to $(\bar{z}_j, u_{-j}) \leq u'$, and thus $(\bar{z}_j, u_{-j}) \leq (\bar{z}_j, u'_{-j})$.

(2) Since $\bar{z} < u$ and $(\bar{z}_j, u_{-j}) \leq u'$ we have $\bar{z}_{-j} < u_{-j} \leq u'_{-j}$. Moreover, with $u' \in B$, we obtain $\bar{z}_j = u'_j$.

□

According to property (1) of Proposition 3.8, the filtering step 7 of Algorithm 3.2 can be replaced by the following step:

$$\begin{aligned} P \leftarrow \{&(\bar{z}_j, u_{-j}) \in P : (\bar{z}_j, u_{-j}) \not\leq (\bar{z}_j, u'_{-j}), \\ &\forall (\bar{z}_j, u'_{-j}) \in P \text{ and } (\bar{z}_j, u_{-j}) \not\leq u', \forall u' \in B\} \end{aligned}$$

which is equivalent to the following formulation:

$$P \leftarrow \{(\bar{z}_j, u_{-j}) \in P : (\bar{z}_j, u_{-j}) \not\leq u', \forall u' \in (A \cup B) \setminus \{u\}\}$$

From property (2) of Proposition 3.8, it is also possible to do fewer dominance tests against the local upper bounds of B.

Overall, Proposition 3.8 shows that it is only required to perform dominance tests between vectors that differ in all but one component. We present these enhancements in Algorithm 3.3, where we split the sets B and P into p disjoint sets, respectively $B_1, \ldots, B_p$ and $P_1, \ldots, P_p$, to stress the by-component filtering step.

Algorithm 3.3: Update procedure of an upper bound set based on redundancy elimination with an enhanced filtering step

input : $U(N)$, $\bar{z}$ – *Set of local upper bounds and new point*
output : $U(N \cup \{\bar{z}\})$

$A \leftarrow \{u \in U(N) : \bar{z} < u\}$ – *Search zones that contain* $\bar{z}$
for $j \in \{1, \ldots, p\}$ **do**
  $B_j \leftarrow \{u \in U(N) : \bar{z}_j = u_j \text{ and } \bar{z}_{-j} < u_{-j}\}$
  $P_j \leftarrow \emptyset$
for $u \in A$ **do**
  for $j \in \{1, \ldots, p\}$ **do**
    $P_j \leftarrow P_j \cup \{(\bar{z}_j, u_{-j})\}$ – *Generate all projections of* $\bar{z}$ *on the local upper bounds of* A
for $j \in \{1, \ldots, p\}$ **do**
  $P_j \leftarrow \{(\bar{z}_j, u_{-j}) \in P_j : (\bar{z}_j, u_{-j}) \not\leq u', \forall u' \in P_j \cup B_j\}$ – *Filter out all redundant points of* P
$U(N \cup \{\bar{z}\}) \leftarrow (U(N) \setminus A) \cup \bigcup_{j=1}^{p} P_j$

While Algorithm 3.3 allows the correct computation of upper bound sets, it requires the iterative filtering for a possibly large number of candidate local upper bounds, which may be computationally expensive. In the next section, we establish structural properties of local upper bounds which yield necessary and sufficient conditions for a candidate local upper bound to become actually a (non-redundant) local upper bound. Then a new approach to the incremental computation of an upper bound set, which avoids the filtering step, is derived.

3.3 Properties of local upper bounds and their efficient computation

In this section, we study some theoretical properties of local upper bounds that yield another approach which, in comparison to the algorithms presented in Section 3.2, avoids the filtering step (namely Steps 8-9 in Algorithm 3.3).

The properties are first presented under a simplifying assumption that no two distinct points, among the points of Z to be considered, share the same value in any dimension. This assumption, denoted "SA" in the remainder, corresponds to what is referred to as a *general position* assumption in computational geometry. It is, however, not realistic for many instances of MOCO problems, that is why we extend the properties under the general case according to which identical component values among distinct points are allowed.

We first illustrate the properties on small examples (Section 3.3.1). Then we detail the properties and derive the new approach (Section 3.3.2).

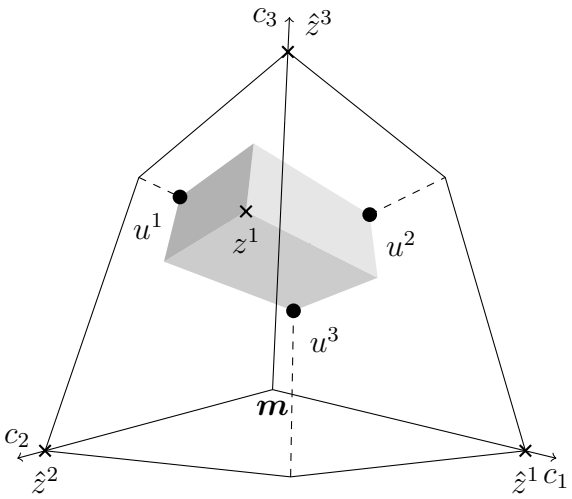
3.3.1 Introductory examples and geometric interpretation

In this section, we give a geometric intuition to the properties that are detailed in the next sections through two example instances in the tri-objective case. First we present an example instance in the SA case (Example 2). Then we discuss the consequences of feasible points having identical component values (Example 3).

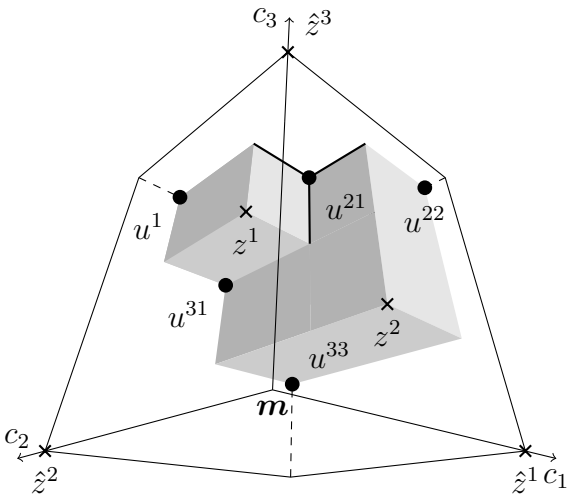
Example 2 (Under SA)**.** We consider a three-dimensional simple instance of our problem which consists of two feasible points : $z^1 = (3, 5, 7)$ and $z^2 = (6, 2, 4)$. Let us apply the incremental algorithm presented in the previous section first on $U(\emptyset) = \{\boldsymbol{M}\}$ and $\bar{z} = z^1$ then on $U(\{z^1\})$ and $\bar{z} = z^2$. At the first iteration, z^1 yields three local upper bounds, namely $u^1 = (3, M, M)$, $u^2 = (M, 5, M)$, and $u^3 = (M, M, 7)$ so that $U(\{z^1\}) = \{u^1, u^2, u^3\}$. Then at the second iteration we consider the three projections of z^2 on the local upper bounds whose associated search zones contain z^2 which are u^2 and u^3. We get $u^{21} = (6, 5, M)$, $u^{22} = (M, 2, M)$, and $u^{23} = (M, 5, 4)$ for u^2, and $u^{31} = (6, M, 7)$, $u^{32} = (M, 2, 7)$, and $u^{33} = (M, M, 4)$ for u^3. Projections u^{23} and u^{32} being redundant since $u^{33} \geq u^{23}$ and $u^{22} \geq u^{32}$, we have $U(\{z^1, z^2\}) = \{u^1, u^{21}, u^{22}, u^{31}, u^{33}\}$.

We represent the situation in Figure 3.2. The feasible points z^1, z^2 are depicted together with their Pareto dominance cones $\{z \in Z : z^i \leqq z\}$, $i = 1, 2$, in gray as well as the local upper bounds. The scene is represented in perspective from point $\boldsymbol{m}$ to point $\boldsymbol{M}$ so that the search zones go towards us.

Now we look at a particular local upper bound, say $u^{21} = (z^2_1, u^2_{-1})$. Consider any point $\bar{z}$ that belongs to the search zone defined by u^{21}. The jth projection of $\bar{z}$ on u^{21} amounts to sliding u^{21} along the half-line $[u^{21}, (\boldsymbol{m}_j, u^{21}_{-j}))$. From Figure 3.2, we can see that if a projection of $\bar{z}$ on u^{21} lies outside any of the three black line segments that start from u^{21},



(a) $N = \{z^1\}$



(b) $N = \{z^1, z^2\}$

Figure 3.2: A 3-dimensional example problem with points under SA

then it will be redundant since it belongs to the closure of another search zone. We can see that these line segments are edges of the union of dominance cones associated to the points of N, plus 3 dummy points $\hat{z}^1 = (M, m, m)$, $\hat{z}^2 = (m, M, m)$ and $\hat{z}^3 = (m, m, M)$. With these dummy points, even local upper bounds located on the boundary of $\hat{Z}$ lie at the intersection of 3 dominance cones. We can now avoid the filtering step (Steps 8-9) of Algorithm 3.3 if, for each local upper bound u, the p edges of the union which are incident to u are known. In the rest of this section, we consider facets of the union of the dominance cones associated to the points of N and to the dummy points $\hat{z}^1, \hat{z}^2, \hat{z}^3$.

We can see that the facets incident to u^{21} are composed of two facets incident to u^2 that are shrunk after the first projection of z^2 and one facet which is a subset of a facet of the dominance cone associated to z^2. So, in order to compute the edges incident to u^{21}, we only have to keep track of the three points that lower bound the facets, namely z^2, z^1, and $\hat{z}^3$. This holds because under SA, a facet is defined by a local upper bound and a single point of N.

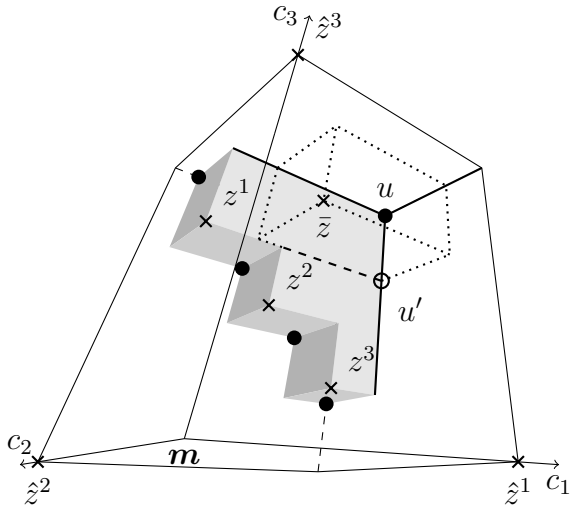


Figure 3.3: A 3-dimensional instance with feasible points having the same value on component 2

Example 3. Consider the 3-dimensional instance represented in Figure 3.3 with three feasible points $z^1 = (2, 7, 7)$, $z^2 = (5, 7, 5)$ and $z^3 = (8, 7, 3)$, which all share the same value on the second coordinate. We look again at facets of the union of all dominance cones associated to the points of N. The local upper bound $u = (M, 7, M)$ is defined by $\hat{z}^1$ on component 1, $\hat{z}^3$ on component 3, and z^1, z^2, and z^3 on component 2. We consider the facet of the union incident to u and orthogonal to the c_2-axis. Similarly to the SA case, we may want to represent this facet by u and a single point defining a lower bound on the c_1 and c_3 values. Since this facet is incident to three feasible points we could define $b = (z^1_1, z^1_2 = z^3_2, z^3_3) = (2, 7, 3)$ (see again Figure 3.3).

However, this information may not be sufficient to avoid future redundancies. Consider for example the point $\bar{z} = (4, 3, 7)$ as depicted in Figure 3.3 together with its Pareto dominance cone (in dotted lines). It satisfies $z_1^1 < \bar{z}_1 < z_1^2$, $\bar{z}_2 < z_2^1 = z_2^2$ and $\bar{z}_3 = z_3^1$. $\bar{z}$ defines among others the local upper bound $u' = (\hat{z}_1^1, z_2^2, \bar{z}_3) = (M, 7, 7)$. Unfortunately, one of the edges incident to u' represented as a dashed line in the figure is limited by an intermediate feasible point, namely z^2. Therefore, it will be necessary in the general case to keep track of all feasible and dummy points that belong to a facet incident to a local upper bound. This is what the sets $Z^j(\cdot)$ are aimed at in Section 3.3.2.

3.3.2 Theoretical properties of local upper bounds and a new incremental approach

According to Step 7 of Algorithm 3.3, all components of a local upper bound u result from previously generated upper bounds for $p-1$ components and, for the remaining component, from the currently added point $\bar{z}$. The initial local upper bound $\boldsymbol{M}$, however, is not defined from any point of Z. In order to make no particular case of the component values inherited from $\boldsymbol{M}$, we extend any stable set of points from Z with the dummy points we introduced in the previous section. Namely, we define the *extension* of N as the set $\hat{N} = N \cup \{\hat{z}^j,\, j = 1, \ldots, p\}$, where

$$\hat{z}^j = (\boldsymbol{M}_j, \boldsymbol{m}_{-j}),\, j = 1, \ldots, p$$

It is not hard to see that $U(\{\hat{z}^j,\, j = 1, \ldots, p\}) = \{\boldsymbol{M}\}$, i.e. the dummy points yield the initial search zone, which implies that for any finite and stable set N of points from Z, we have $U(\hat{N}) = U(N)$.

Using dummy points, we now have that any component value of a local upper bound is defined by a point of $\hat{N}$.

Observe that a dummy point $\hat{z}^j$ can only define the jth component of any local upper bound, which is M. Indeed since no point from Z is lower than or equal to m on any component, m cannot be a component value of a local upper bound. Therefore, and since M is unique in the component values of a dummy point, $\hat{z}^j$ is the only dummy point that can define component j.

The following proposition gives a useful property of those points that define each component of a local upper bound.

Proposition 3.9. *For any local upper bound $u \in U(N)$ and $j \in \{1, \ldots, p\}$, there exists $z \in \hat{N}$ such that $z_j = u_j$ and $z_{-j} < u_{-j}$.*

Proof. If $u_j = M$, then the dummy point $\hat{z}^j \in \hat{N}$ satisfies the required conditions.

Otherwise and since $\hat{N}$ is a finite set, there exists an $\varepsilon > 0$ sufficiently small such that no point of $\hat{N}$ has its jth component value in the interval $(u_j, u_j + \varepsilon)$. Let $u' = (u_j + \varepsilon, u_{-j})$.

According to Proposition 3.6, since $u' \in \hat{Z}$ and $u' \geq u$, there exists a $z \in N$ such that (i) $z < u'$ and (ii) $z \not< u$. It follows from (i) that we have $z_{-j} < u_{-j}$, which imposes $z_j \geq u_j$ from (ii). From the choice of ε, we therefore have $z_j = u_j$. □

In the following we define two notations for those points that define local upper bounds, for the general case and for the SA case, respectively.

Definition 3.10. For any local upper bound $u \in U(N)$, we denote by $Z^j(u) = \{z \in \hat{N} : z_j = u_j \text{ and } z_{-j} < u_{-j}\}$ the set of defining points of u for component j, $j = 1, \ldots, p$.

In the SA case, the unique defining point of u for component j is denoted $z^j(u)$.

Using Proposition 3.9, we can now precisely characterize the projections which are kept in the set P after the filtering step of Algorithm 3.3. We first consider the SA case.

Theorem 3.11 (Simplifying assumption)**.** *Let $\bar{z}$ be a point of Z that is nondominated with respect to N and such that the points in $N \cup \{\bar{z}\}$ satisfy SA. Consider a local upper bound $u \in U(N)$ such that $\bar{z} < u$. Let $z_j^{\max}(u) = \max_{k \neq j}\{z_j^k(u)\}$.*

Then, for any $j \in \{1, \ldots, p\}$, $(\bar{z}_j, u_{-j})$ is a local upper bound of $U(N \cup \{\bar{z}\})$ if and only if $\bar{z}_j > z_j^{\max}(u)$.

Proof. Let $u \in U(N)$ and $\bar{z} \in Z \setminus N$ be a point not dominated by any point of N such that $\bar{z} < u$.

($\Rightarrow$) Suppose that $\bar{u} = (\bar{z}_j, u_{-j})$ is a local upper bound in $U(N \cup \{\bar{z}\})$ and let $z_j^{\max}(u) = z_j^k(u)$ for some point $z^k(u) \in \hat{N}$, such that $z_k^k(u) = u_k$, $k \neq j$. Therefore, $z_k^k(u) = \bar{u}_k$ and, from SA, no other point of $\hat{N}$ equals $\bar{u}_k$ on its kth component. Thus from Proposition 3.9, we have $z_{-k}^k(u) < \bar{u}_{-k}$, which implies $z_j^k(u) = z_j^{\max}(u) < \bar{u}_j = \bar{z}_j$.

($\Leftarrow$) Assume that for a given $j \in \{1, \ldots, p\}$, $\bar{z}_j > z_j^{\max}(u)$. Suppose, to the contrary, that $(\bar{z}_j, u_{-j})$ is not a local upper bound for $N \cup \{\bar{z}\}$, that is, it dominates a local upper bound of $U(N \cup \{\bar{z}\})$. Hence from Proposition 3.8, there exists $u' \in U(N)$ such that $(\bar{z}_j, u_{-j}) \leq (\bar{z}_j, u'_{-j})$ (note that in the SA case, the set B defined in Algorithm 3.2 is empty). Then, we have $u_{-j} \leq u'_{-j}$, which implies $u_j > u'_j$ and $u_k < u'_k$ for some $k \neq j$. Let $z^k(u) \in \hat{N}$ be the point that defines the kth component u_k of u. From Proposition 3.9, we have $z_{-k}^k(u) < u_{-k}$. Thus, since $k \neq j$, we have $z_{-j}^k(u) < u'_{-j}$ but since u' is a local upper bound, we must have $z_j^k(u) \geq u'_j$ (otherwise $z^k(u) < u'$). Hence, $z_j^{\max}(u) \geq z_j^k(u) \geq u'_j$. Since we have both $\bar{z}_j > z_j^{\max}(u)$ and $\bar{z}_j < u'_j$, we obtain a contradiction: $\bar{z}_j < u'_j \leq z_j^{\max}(u) < \bar{z}_j$.

□

Let us illustrate this theorem on the first example instance of Section 3.3.1.

Example 2 (continued)**.** Consider the situation in Figure 3.2a with $N = \{z^1\}$, where $z^1 = (3, 5, 7)$. The points that define the local upper bounds of $U(N)$, namely $u^1 = (3, M, M)$, $u^2 = (M, 5, M)$, and $u^3 = (M, M, 7)$, are:

$$\begin{array}{lll} z^1(u^1) = z^1 & z^2(u^1) = \hat{z}^2 & z^3(u^1) = \hat{z}^3 \\ z^1(u^2) = \hat{z}^1 & z^2(u^2) = z^1 & z^3(u^2) = \hat{z}^3 \\ z^1(u^3) = \hat{z}^1 & z^2(u^3) = \hat{z}^2 & z^3(u^3) = z^1 \end{array}$$

and $z^{\max}(u^1) = (m, 5, 7)$, $z^{\max}(u^2) = (3, m, 7)$, and $z^{\max}(u^3) = (3, 5, m)$.

The point $z^2 = (6, 2, 4)$ strictly dominates u^2 and u^3 and we have:

$$\begin{array}{lll} z^2_1 > z^{\max}_1(u^2) & z^2_2 > z^{\max}_2(u^2) & z^2_3 \leq z^{\max}_3(u^2) \\ z^2_1 > z^{\max}_1(u^3) & z^2_2 \leq z^{\max}_2(u^3) & z^2_3 > z^{\max}_3(u^3) \end{array}$$

thus we obtain again the four new local upper bounds $u^{21} = (z^2_1, u^2_{-1})$, $u^{22} = (z^2_2, u^2_{-2})$, $u^{31} = (z^2_1, u^3_{-1})$, and $u^{33} = (z^2_3, u^3_{-3})$.

According to Theorem 3.11, we can avoid the filtering step of Algorithm 3.3 if we keep track of the p points that define each local upper bound and only generate the projections of $\bar{z}$ that satisfy the conditions of Theorem 3.11. The corresponding algorithm is detailed in Algorithm 3.4.

Note that each component of the vector $z^{\max}(u)$ for a given local upper bound u will be used at most once in all iterations of Algorithm 3.4. That is why it is computed only before its use, namely at Step 5. Moreover, this vector is not sufficient to compute the vector $z^{\max}(u^j)$ associated to a local upper bound u^j defined from u. Indeed, as the following example shows, it is required to keep track of all points that define the component values of u^j, as is done in Steps 8-10.

Example 2 (continued)**.** Consider a new point $z^3 = (4, 4, 2)$ and the local upper bound $u^{21} = (6, 5, M)$ with $z^{\max}(u^{21}) = (3, 2, 7)$, stemming from $z^1(u^{21}) = z^2 = (6, 2, 4)$, $z^2(u^{21}) = z^1 = (3, 5, 7)$, and $z^3(u^{21}) = \hat{z}^3 = (m, m, M)$.

We have $z^3 < u^{21}$ and $z^3_2 \geq z^{\max}_2(u^{21})$, thus $u^{212} = (z^3_2, u^{21}_{-2})$ is a local upper bound of $U(\{z^1, z^2, z^3\})$. Since $z^1(u^{212}) = z^1(u^{21})$, $z^2(u^{212}) = z^3$, and $z^3(u^{212}) = z^3(u^{21})$, we have $z^{\max}(u^{212}) = (4, 2, 4)$. As we can see, the last component value of $z^{\max}(u^{212})$, which comes from $z^1(u^{212}) = z^2$, cannot be obtained from $z^{\max}(u^{21})$ or z^3.

In the general case, Theorem 3.11 is modified as follows:

Theorem 3.12. *Let N be a finite and stable set of points of Z, and let $\bar{z}$ be a point of Z that is nondominated with respect to N. Consider a local upper bound $u \in U(N)$ such that $\bar{z} < u$. Let $z^{\max}_j(u) = \max_{k \neq j} \min\{z_j : z \in Z^k(u)\}$.*

Then, for any $j \in \{1, \ldots, p\}$, $(\bar{z}_j, u_{-j})$ is a local upper bound of $U(N \cup \{\bar{z}\})$ if and only if $\bar{z}_j > z^{\max}_j(u)$.

Algorithm 3.4: Update procedure of an upper bound set based on the avoidance of redundancies : *SA case*

input : $U(N)$ together with $z^j(u), \forall j \in \{1,\dots,p\}, u \in U(N)$ – *Set of local upper bounds and associated defining points*
input : $\bar{z}$ – *New point*
output : $U(N \cup \{\bar{z}\})$

$A \leftarrow \{u \in U(N) : \bar{z} < u\}$ – *Search zones that contain* $\bar{z}$
$P \leftarrow \emptyset$
for $u \in A$ **do**
  for $j \in \{1,\dots,p\}$ **do**
    $z_j^{\max}(u) \leftarrow \max_{k \neq j}\{z_j^k\}$
    – *Check for the condition of Theorem 3.11* –
    if $\bar{z}_j > z_j^{\max}(u)$ **then**
      – *Let* $u^j = (\bar{z}_j, u_{-j})$
      $P \leftarrow P \cup \{u^j\}$
      $z^j(u^j) \leftarrow \bar{z}$
      for $k \in \{1,\dots,p\} \setminus \{j\}$ **do**
        $z^k(u^j) \leftarrow z^k(u)$
$U(N \cup \{\bar{z}\}) \leftarrow (U(N) \setminus A) \cup P$

Proof. Let $u \in U(N)$ and $\bar{z} \in Z \setminus N$ be a point not dominated by any point of N such that $\bar{z} < u$.

($\Rightarrow$) Suppose that $\bar{u} = (\bar{z}_j, u_{-j})$ is a local upper bound in $U(N \cup \{\bar{z}\})$ and to the contrary $\bar{z}_j \leq z_j^{\max}(u)$. Then, there is $k \neq j$ such that $\bar{z}_j \leq z_j$ for all $z \in Z^k(u)$. Since $\bar{u} \leq u$ and $\bar{u}_k = u_k$, it holds that $Z^k(\bar{u}) \subset Z^k(u)$ but for any $z \in Z^k(u)$, $z_j \not< \bar{z}_j = \bar{u}_j$. Hence, $Z^k(\bar{u}) = \emptyset$ which contradicts Proposition 3.9.

($\Leftarrow$) Assume that for a given $j \in \{1,\dots,p\}$, $\bar{z}_j > z_j^{\max}(u)$. Suppose, to the contrary, that $(\bar{z}_j, u_{-j})$ is not a local upper bound for $N \cup \{\bar{z}\}$, that is, it dominates a local upper bound of $U(N \cup \{\bar{z}\})$. Hence from Proposition 3.8, there exists $u' \in U(N)$, $(\bar{z}_j, u_{-j}) \leq (\bar{z}_j, u'_{-j})$ (possibly with $\bar{z}_j = u'_j$).

Then we have $u_{-j} \leq u'_{-j}$ which implies $u_j > u'_j$ and $u_k < u'_k$ for some $k \neq j$. From Proposition 3.9, the set $Z^k(u)$ is non-empty. For any $z \in Z^k(u)$, we have $z_{-k} < u_{-k}$ and thus, since $k \neq j$, $z_{-j} < u'_{-j}$ but since u' is a local upper bound, we must have $z_j \geq u'_j$ (otherwise $z < u'$). Hence, there is a $z \in Z^k(u)$ such that $z_j^{\max}(u) \geq z_j \geq u'_j$. Since we have both $\bar{z}_j > z_j^{\max}(u)$ and $\bar{z}_j \leq u'_j$, we obtain a contradiction: $\bar{z}_j \leq u'_j \leq z_j^{\max}(u) < \bar{z}_j$.

□

We illustrate the general case on the second example instance of Section 3.3.1.

Example 3 (continued)**.** In Figure 3.3, we consider the situation with $N = \{z^1, z^2, z^3\}$ where $z^1 = (2, 7, 7)$, $z^2 = (5, 7, 5)$ and $z^3 = (8, 7, 3)$. We only look at the local upper bound $u = (M, 7, M)$. We have $Z^1(u) = \{\hat{z}^1\}$, $Z^2(u) = \{z^1, z^2, z^3\}$, and $Z^3(u) = \{\hat{z}^3\}$. Thus $z^{\max}(u) = (2, m, 3)$. The projections of a point $\bar{z}$ that strictly dominates u will be kept as non-redundant local upper bounds depending on the comparisons between the component values of $\bar{z}$ and $z^{\max}(u)$ only.

Algorithm 3.5 presents the update procedure in the general case. The initialization is done with $U(\emptyset) = \{\boldsymbol{M}\}$ and $Z^j(\boldsymbol{M}) = \{\hat{z}^j\}$, $j = 1, \ldots, p$.

Algorithm 3.5: Update procedure of an upper bound set based on the avoidance of redundancies : *general case*

input : $U(N)$ together with $Z^j(u)$ for all $j \in \{1, \ldots, p\}$, $u \in U(N)$ – *Set of local upper bounds and associated defining points*
input : $\bar{z}$ – *New point*
output : $U(N \cup \{\bar{z}\})$

$A \leftarrow \{u \in U(N) : \bar{z} < u\}$ – *Search zones that contain* $\bar{z}$
$P \leftarrow \emptyset$
– *Update sets* $Z^j(u)$ *when* $\bar{z}$ *satisfies the conditions of Proposition 3.9* –
for $u \in U(N)$ *and* $j \in \{1, \ldots, p\}$ **such that** $\bar{z}_j = u_j$ *and* $\bar{z}_{-j} < u_{-j}$ **do**
  $Z^j(u) \leftarrow Z^j(u) \cup \{\bar{z}\}$
for $u \in A$ **do**
  for $j \in \{1, \ldots, p\}$ **do**
    $z_j^{\max}(u) \leftarrow \max_{k \neq j} \min\{z_j : z \in Z^k(u)\}$
    – *Check for the condition of Theorem 3.12* –
    if $\bar{z}_j > z_j^{\max}(u)$ **then**
      – *Let* $u^j = (\bar{z}_j, u_{-j})$
      $P \leftarrow P \cup \{u^j\}$
      $Z^j(u^j) \leftarrow \{\bar{z}\}$
      for $k \in \{1, \ldots, p\} \setminus \{j\}$ **do**
        $Z^k(u^j) \leftarrow \{z \in Z^k(u) : z_j < \bar{z}_j\}$
$U(N \cup \{\bar{z}\}) \leftarrow (U(N) \setminus A) \cup P$

3.4 Complexity and computational experiments

In Sections 3.2 and 3.3, we described two incremental approaches for the update of an upper bound set. In Section 3.2, the approach is based on redundancy elimination (RE) among

local upper bounds, while in Section 3.3 it is based on redundancy avoidance (RA) with respect to local upper bounds.

We first report upper bounds on the total number of local upper bounds associated to a discrete set of points N. Then we study the complexities of the RE and RA approaches. Finally, we present some computational experiments that compare these approaches.

3.4.1 Tight upper bound on the number of local upper bounds

None of the incremental algorithms proposed in the literature, even in the SA case, make it possible to directly derive a non-trivial upper bound on the size of any upper bound set $U(N)$ for $p \geq 4$.

For $p = 2$, the number of local upper bounds is clearly $|N| + 1$ (see Example 1). For $p = 3$ we recall that Dächert and Klamroth (2014) showed that it is bounded above by $2|N| + 1$ and is exactly $2|N| + 1$ in the SA case.

For an arbitrary $p \geq 2$, Kaplan et al. (2008) provide a tight upper bound on the size of $U(N)$. Following Boissonnat et al. (1998) who studied the complexity of a union of axis-parallel hypercubes, they show that the number of maximal empty orthants with respect to a stable set N is $O(|N|^{\lfloor \frac{p}{2} \rfloor})$. They also provide an instance for which this number is $\Omega(|N|^{\lfloor \frac{p}{2} \rfloor})$. Therefore, and recalling that maximal empty orthants are in one-to-one correspondence with local upper bounds, $O(|N|^{\lfloor \frac{p}{2} \rfloor})$ is a tight upper bound on the total number of local upper bounds associated to a stable set N.

3.4.2 Worst-case complexities of the algorithms

In this section, we analyze the worst case behavior of the two approaches. We consider the dimension p of the problem as a fixed parameter. The reference algorithm for the RE approach will be Algorithm 3.3 while the reference algorithm for the RA approach will be Algorithm 3.4 in the SA case, and Algorithm 3.5 in the general case.

Common steps of both algorithms

In the SA case, both approaches first compute the set A of local upper bounds that contain $\bar{z}$. This amounts to $|U(N)|$ dominance tests if $U(N)$ is stored as a simple linked list. If A is small compared to $U(N)$, it is possible to reduce the complexity of these operations. Actually, since the elements of A are those local upper bounds located in the hyperrectangle $\prod_{j=1}^{p}(\bar{z}_j, M)$, they can be obtained by an orthogonal range query on the set $U(N)$ (see Berg et al. (2008), Chapter 5). In the case $p = 2$, $U(N)$ can be efficiently stored in a simple balanced binary search tree. For $p \geq 3$, as in the case of the algorithm of Kaplan et al. (2008) $U(N)$ can be stored in a dynamic p-dimensional range tree, as proposed e.g. in Willard and Lueker (1985), which allows insertions and deletions in $O(\log^p |U(N)|)$ time and orthogonal range queries in $O(\log^p |U(N)| + |A|)$ time. We note that *augmented*

dynamic range trees (Mehlhorn and Näher (1990), Theorem 8) lower the "log" factors to $\log^{p-1}|U(N)|\log\log|U(N)|$.

Remaining steps

We assume that $p \geq 3$ since it can be easily seen that both approaches operate identically in the case $p = 2$. Both approaches consider $p|A|$ candidate local upper bounds.

We first consider the SA case. We focus on the operations on which Algorithm 3.3 (RE approach) and Algorithm 3.4 (RA approach) differ. They correspond to Steps 5-9 (Algorithm 3.3) and Steps 3-10 (Algorithm 3.4), and respectively involve sets P_j, $j = 1, \ldots, p$, and P.

Proposition 3.13. *The worst-case complexity of Steps 5-9 in Algorithm 3.3 is bounded by* $O(|A|^2)$.

Proof. The complexity of these steps is dominated by the filtering (Steps 8-9) of each P_j, $j = 1, \ldots, p$, where $|P_j| = |A|$, therefore the total time is $O(|A|^2)$. □

This can be reduced to $O(|A|\log|A|)$ in the case $p = 2, 3$ (Kung et al. (1975)) and $O(|A|\log^{p-3}|A|\log\log|A|)$ in the case $p \geq 4$ (Gabow et al. (1984)) using some specialized algorithms.

Proposition 3.14. *The worst-case complexity of Steps 3-10 in Algorithm 3.4 is bounded by* $O(|A|)$.

Proof. In Algorithm 3.4, no additional dominance test is performed with the local upper bounds of P, but the values $z_j^{\max}(u)$ need to be computed just before they are needed, each of which takes constant time. Also the references to the p points that define each local upper bound have to be updated which takes constant time for each new upper bound. The total time of these operations is thus $O(|A|)$. □

In the general case, the number of local upper bounds against which candidate local upper bounds have to be checked for dominance in the RE approach just grows by an additional $|B|$. In the RA approach adapted to the general case, namely Algorithm 3.5, it is possible that $|N|$ points have to be considered in a set $Z^k(u)$ at Step 12. This leads to an upper bound on the complexity of $O(|N||A|)$ in Algorithm 3.5.

In practice, however, the size of the sets $Z^k(u)$ is rather small depending on how many points in N share the same component values. Note that according to Boissonnat et al. (1998), an alternative approach would be to slightly shift those points in N that do not satisfy SA such that the resulting set satisfies SA. Then Algorithm 3.4 can be applied, yielding a complexity of $O(|A|)$. Similarly, ties in the comparisons of any jth component values for points z^k and z^l could be resolved by a lexicographic comparison "$<_{\text{lex}}$" where $z_j^k <_{\text{lex}} z_j^l$ if $z_j^k < z_j^l$ or if $z_j^k = z_j^l$ and $k < l$, which would replace the natural comparison

"<" between reals (and similarly for ">") in Algorithm 3.4. However these approaches yield redundant search zones that, in the context of Algorithm 3.1, induce redundant solver calls.

3.4.3 Experimental comparison of the algorithms

In this section we investigate the behaviour of the RE and RA approaches on random instances.

Experimental setup

We implemented Algorithm 3.3 for the RE approach and Algorithm 3.5 for the RA approach. Both algorithms were implemented in C. The experiments were run on a workstation equipped with an Intel Core i7-3840QM CPU at 2.80GHz with 8MB cache and 32GB RAM. For both algorithms, we considered the version that does not require SA, since the assumption cannot be made in most applications.

As test instances, we generated random stable sets of points N. In order to obtain a new point in the random stable set being generated, we uniformly draw from the integer set $[1, K]^p$ and reject the points that are dominated by or dominate any of the previous points. We draw without or with replacement in $[1, K]$, respectively, to obtain points satisfying SA or not. In the general case, the parameter K controls to what extent objective values are shared among feasible points. In the SA case, K is just set to a very large integer. Since in both cases the distribution of each point is conditioned by the requirement that it is neither dominated by nor dominates any previously generated point, the generated points are eventually randomly reordered.

We considered instances for $p = 3, 4, 5, 6$ having 100 000, 50 000, 25 000, and 5 000 points, respectively. We generated instances under SA and also with possible identical component values. In the general case, we set K so as to obtain $\frac{|N|}{K} = 5, 10$. The plots we made in the SA case were obtained by recording intermediate results every 500 points for $p = 3, 4, 5$ and every 100 points for $p = 6$. We also considered a pathological instance type in the general case with $p = 6$, $|N| = 10\,000$ and $K = 10$.

We drawn 10 instances of each type and the output results were averaged over the 10 runs carried out for each instance type.

Observations on the SA instances

We provided above a theoretical tight upper bound on the number of local upper bounds in SA instances. Now we consider the empirical number of local upper bounds observed in our test instances for $p = 4, 5, 6$ (since the theoretical upper bound is tight for $p = 3$). The results, which can be obtained by any of the two approaches, are reported on Figure 3.4. According to Figure 3.4, it seems that on such random instances, the number of local upper bounds grows approximately linearly in the number of points. Kaplan et al. (2008)

showed that the number of maximal empty axis-parallel boxes in a set of n points drawn uniformly and independently from $[0,1]^p$ is $O(n \log^{p-1} n)$, therefore the growth observed in our experiments may be superlinear. However, the distribution of our points is not the same since we discard points that dominate or are dominated by previously drawn points and the bound of Kaplan et al. (2008) does not count only maximal empty orthants.

Observing from Figure 3.4 the apparently linear relation between $|N|$ and $|U(N)|$, we performed a simple linear regression. We obtained the following slopes for the fitted lines: 6.524 for $p = 4$, 31.86 for $p = 5$, and 165.9 for $p = 6$. This gives an insight on the increase in the number of local upper bounds induced by the consideration of a new point in the search region, i.e. the average $|U(N \cup \{\bar{z}\})| - |U(N)|$.

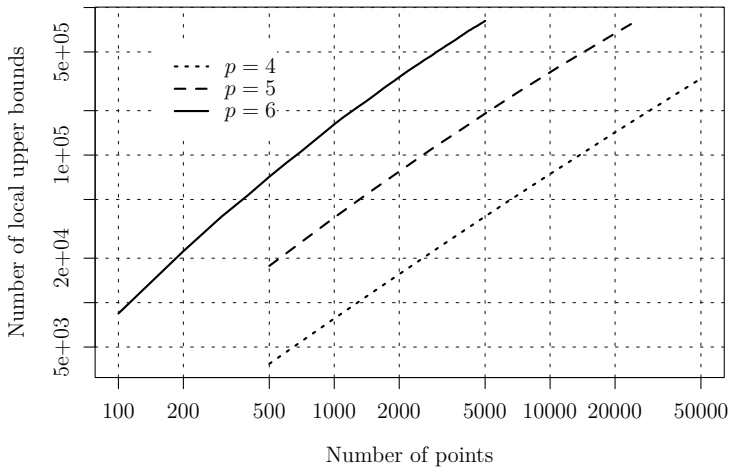


Figure 3.4: Number of local upper bounds on SA instances (logarithmic scales for both axes)

We also computed the average number of search zones that contain the current point (namely $|A|$) between two observations. Since these values do not vary much on the tested instance, we provide the averages over all instances of all sizes: 3.999 for $p = 3$, 21.56 for $p = 4$, 141.67 for $p = 5$, and 735.9 for $p = 6$.

From the average $|U(N \cup \{\bar{z}\})| - |U(N)|$ and $|A|$, we compute the ratio $\frac{|U(N \cup \{\bar{z}\})| - |U(N)|}{|A|}$. We obtain 0.5001 for $p = 3$, 0.3025 for $p = 4$, 0.2249 for $p = 5$, and 0.2255 for $p = 6$. This indicates that the number of additional search zones induced by each search zone that contains the current feasible point remains small.

Comparison of the algorithms

We first provide raw computation times in Table 3.1, showing the performance of the RE and RA approaches on SA and on general instances.

	RE approach				RA approach			
$\lvert N\rvert$ \ p	3	4	5	6	3	4	5	6
5 000	0.232	1.09	26.3	614.0	0.289	2.18	35.2	179.0
25 000	6.93	93.4	830.0	-	16.6	154.0	951.0	-
50 000	44.6	509.0	-	-	122.0	673.0	-	-
100 000	387.0	-	-	-	664.0	-	-	-

(a) Simplifying assumption

	RE approach				RA approach			
$\lvert N\rvert$ \ p	3	4	5	6	3	4	5	6
5 000	0.192	0.882	13.4	530.0	0.24	1.15	18.7	166.0
25 000	6.18	68.3	767.0	-	9.53	112.0	862.0	-
50 000	36.0	463.0	-	-	77.0	582.0	-	-
100 000	339.0	-	-	-	498.0	-	-	-

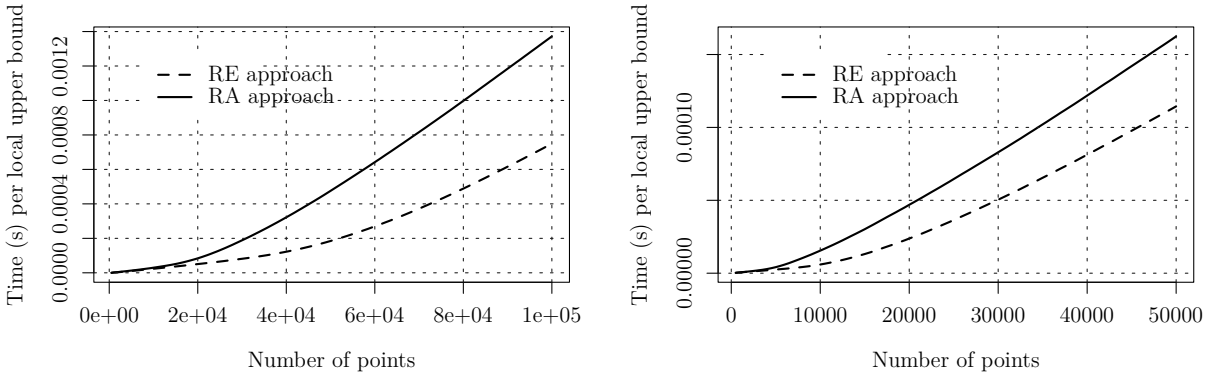
(b) No simplifying assumption, $\frac{|N|}{K} = 5$

	RE approach				RA approach			
$\lvert N\rvert$ \ p	3	4	5	6	3	4	5	6
5 000	0.167	0.772	12.1	447.0	0.218	0.978	16.7	150.0
25 000	5.99	65.6	754.0	-	7.84	107.0	839.0	-
50 000	32.8	447.0	-	-	69.2	564.0	-	-
100 000	325.0	-	-	-	459.0	-	-	-

(c) No simplifying assumption, $\frac{|N|}{K} = 10$

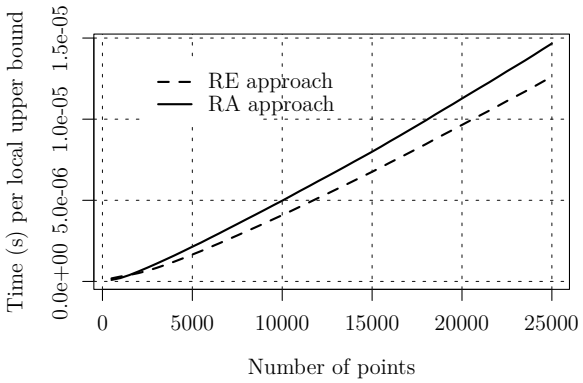
Table 3.1: Average computation times (in seconds) for both approaches

Since the computation times of the algorithms we consider to generate $U(N \cup \{\bar{z}\})$ are both $\Omega(|U(N)|)$, we also present normalized computation times. Figure 3.5 shows running times divided by $|U(N)|$.

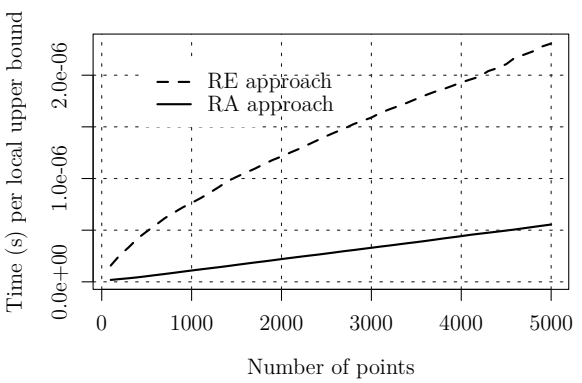


(a) $p = 3$

(b) $p = 4$



(c) $p = 5$



(d) $p = 6$

Figure 3.5: Comparison of the normalized running times of the two algorithms in the SA case

According to these results, the RE approach remains the best one in terms of computation time for $p = 3, 4, 5$, the values being rather close in the case $p = 5$. The RA approach however outperforms the RE approach for $p = 6$. These observations hold for SA and general case instances but the gaps between the relative efficiency of the approaches are larger on SA instances. Besides, additional computational experiments on SA instances with $p = 7, 8$ showed that the RA approach performs even better above $p = 6$. Namely, we obtained the following average computation times (RE time in seconds, RA time in seconds): (161.69, 5.046) for $p = 7$, $n = 500$, (1 240.11, 27.458) for $p = 7$, $n = 1\,000$, (24.06,

0.348) for $p = 8$, $n = 125$, (423.15, 2.330) for $p = 8$, $n = 250$.

We also observed in our experiments that, even starting from $p \geq 3$, a little fewer component comparisons are made in the RA approach than in the RE approach. Finally, we ran the implementations under Cachegrind, a CPU caches profiling tool. We observed for the RA approach a larger use of the slowest caches, L2 and L3, than for the RE approach. This, together with the smaller average $|A|$ observed on low dimensional instances, explains why the implementation of the RA approach performs worse than the one of the RE approach for $p \leq 5$.

To observe the effect of highly duplicated component values among distinct points, we also tested the approaches on instances of points with $p = 6$, $|N| = 10\,000$, and $K = 10$. The average $|U(N)|$ and $|A|$ are much smaller than in the SA case, being respectively 14 228.4 and 33.44. Due to the fact that many points share the same component values, the sets $Z^k(u)$ in Algorithm,3.5 can grow significantly, reaching the maximum value of 1 109.7, averaged on the test instances. Therefore, the computation time of the RA approach is a little longer than the one of the RE approach (1.95 against 1.51 seconds).

Conclusions

We addressed in this chapter the problem of representing the search region in MOO. The concept itself is used in numerous approaches to compute the nondominated set. We provided several equivalent definitions of the search region. Local upper bounds induce a decomposition of the search region into search zones. We reviewed possible uses of this decomposition to enumerate all nondominated points of an MOO problem. We presented two incremental approaches to compute the local upper bounds that represent a search region, respectively based on "redundancy elimination (RE)" and "redundancy avoidance (RA)". The first encompasses an already known algorithm for which we proposed some enhancements to its filtering step. The second is derived from theoretical properties of local upper bounds we studied and avoids the filtering step of the former. Finally, we considered the complexity of the representation of the search region by local upper bounds and gave some insights into the theoretical complexities and the practical efficiencies of the two incremental approaches. In particular, we showed that the RA approach developed in this chapter performs better than the RE approach starting from dimension 6 on instances where the objective ranges are not too small.

The future work directions are numerous. Although the RA approach is practically less efficient than the RE approach in low dimensions, it maintains, contrary to the latter, a relation between feasible points and local upper bounds. This makes it possible to define a neighborhood between local upper bounds, as in Dächert and Klamroth (2014) in the case $p = 3$, that can be exploited in order to update the search region more efficiently when a search zone containing the new feasible point is known. Derivatives of the concept of search

region defined in this chapter could also be considered. Actually, we made no assumption on the feasible points that define the search region, apart from the requirement that they constitute a stable set of points. If e.g. the feasible points are optimal with respect to one objective function, some search zones may be discarded. Note also that the search zones that are defined in this chapter are bounded below by the same point $\boldsymbol{m}$. It may be interesting to bound below each search zone using some *local lower bounds* such that the union of the corresponding restricted search zones still contains all unknown nondominated points.

Chapter 4

Hybrid ranking and branch and bound algorithm for solving multi-objective combinatorial optimization problems

This chapter is concerned with generic approaches for solving MOCO problems. We propose a flexible method primarily aimed at the generation of the nondominated set of a MOCO problem, which is not limited to the bi-objective case. We first propose an approach which relies on the weighted sum technique and on the use of a ranking algorithm. Then this approach is integrated into a branching scheme. We also propose two approximation versions of our hybrid strategy which provide an a priori guarantee on the quality of the output. The proposed generic method is instantiated on the Minimum Spanning Tree problem and computational experiments that show the practical efficiency of our method are reported for several instance types.

Contents

4.1 A ranking strategy . . . 81
4.1.1 Computation of an initial set of points and a weight vector set . . . 82
4.1.2 Association between weight vectors and search zones . . . 83
4.1.3 Exploitation of a ranking direction to explore a group of search zones 84
4.1.4 Organizing the whole search . . . 85
4.2 Integration of the ranking strategy in a branching tree . . . 87
4.2.1 Branching scheme . . . 87
4.2.2 Bounding procedure . . . 88
4.2.3 Stopping criterion . . . 90
4.2.4 Choice of a branching item . . . 90
4.3 Computation of a reduced $(1+\varepsilon)$-approximation . . . 91
4.3.1 Tolerance in the weighted sum test . . . 91
4.3.2 Case of ε-dominated points . . . 91
4.4 Instantiation on the Minimum Spanning Tree problem and implementation issues . . . 93
4.4.1 Ranking spanning trees . . . 93
4.4.2 Computation of an initial solution pool through neighborhood search 94
4.4.3 Preprocessing search nodes . . . 94
4.4.4 Update of the search region . . . 96
Testing membership to the search region • Updating upper bound sets
4.5 Computational experiments . . . 97
4.5.1 Experimental setup . . . 98
Instances
4.5.2 Results . . . 99
General statistics on instances • Comparison of strategies • Opportunity of neighborhood search • Approximation in the ranking strategy

Introduction

As seen in Chapter 1, most approaches to the solution of MOCO problems are either based on implicit enumeration or on the exploration of the search region by iteratively searching in sets of zones. Several computational comparisons of MOBB and two-phase methods, which are representatives of the two general approaches we mentioned, appeared in the literature. To our knowledge, the most recent comparison not limited to the bi-objective case concerns MOBB and ranking-based two-phase methods for the KNAPSACK problem and can be found in Jorge (2010). According to the author, the MOBB approach is dominated by the two-phase method for the KNAPSACK problem. It is well-known however, that ranking approaches encounter difficulties with instances having many nondominated points that are far from being supported.

Our aim is therefore to combine the two approaches into a single method. The foundations of this method are the use of an underlying search tree, the evaluation of search nodes using optimal solutions to weighted sums of the objectives, and the use of a ranking algorithm in order to obtain more extensions from each search node. The proposed approach is therefore a hybrid strategy which behaves at one end as a ranking approach and at the other end as a branch and bound algorithm. Our method is instantiated and tested on the MOST problem.

We assume in the presentation that the decision space of the underlying MOCO problem is 2^E, where E is a set of m items to be taken or not in a solution. The feasible set X is therefore a subset of 2^E.

We chose to first present a ranking strategy (Section 4.1), which is later integrated at each search node of a branching tree (Section 4.2). In Section 4.3 we show how our method can be adapted to generate an ε-approximation of the nondominated set. The instantiation of the proposed method on the MINIMUM SPANNING TREE problem is presented in Section 4.4 as well as some important implementation details. Finally, computational experiments which support our approach are discussed in Section 4.5.

4.1 A ranking strategy

Ranking algorithms have been used several times in MOO and particularly in phase two of the two-phase method. In phase one of the two-phase method, the set Y_{extr} of all extreme nondominated points is computed and also weight vectors associated to the facets of $\text{conv}(Y)_{\leqq}$. The ranking strategy we propose does not necessarily compute Y_{extr} at the beginning. We propose, especially for $p \geq 3$, to compute a subset of Y_{supp} which is possibly augmented by points obtained from a heuristic. Observe that these points do not need to be supported nondominated points. Their role is to refine the initial search region

We present a general ranking strategy in Algorithm 4.1[1] whose main components are detailed in the following subsections.

Algorithm 4.1: Ranking procedure

input : instance data, $\boldsymbol{M}$ – *$\boldsymbol{M}$ is strictly dominated by any feasible point*
output : N – *the nondominated set*

```
1  N ← ∅; U(N) ← {M}    – Initialization of the output and the initial upper bound set
2  Λ ← computeWeightVectorSet()    – Computation of a set of weight vectors
3  for λ ∈ Λ do
4      z^{*λ} ← arg min{λ · z : z ∈ Y}
5      updateSearchRegion(z^{*λ}, ↕N, ↕U(N))
6  Λ' ← Λ
7  while Λ' ≠ ∅ do
8      Select λ ∈ Λ'; Λ' ← Λ'\{λ}
       – Association of search zones from U(N) to a weight vector λ
9      U_λ ← associateSearchZones(U(N), Λ', λ)
10     while U_λ ≠ ∅ do
11         z ← computeNextPoint(λ)
12         updateSearchRegion(z, ↕N, ↕U(N))
13         if ubsTest(λ, z, U_λ) then
14             U(N) ← U(N) \ U_λ
15             U_λ ← ∅
```

4.1.1 Computation of an initial set of points and a weight vector set

We presented in Section 1.2.2 several approaches to compute the set Y_{extr} of all extreme nondominated points of a MOCO problem, which also compute weight vectors associated to the facets of $\text{conv}(Y)_{\geqq}$. In the bi-objective case, this is efficiently done using the algorithm of Aneja and Nair (1979). We stressed that the general multi-objective case raises several difficulties. Additionally, it is well-known that the number of facets of the convex hull of k points of dimension p is $O(k^{\lfloor \frac{p}{2} \rfloor})$ (see e.g. Preparata and Shamos, 1985). Thus, for $p \geq 4$, the number of supporting hyperplanes of $\text{conv}(Y)_{\geqq}$ could no longer be linear in the number of extreme nondominated points.

An alternative approach is the a priori determination of a weight vector set. Borges and Hansen (2002) propose to rely on a *maximally dispersed set* of weight vectors for

[1] In the presentation of the algorithms in pseudo-code, the modified arguments of procedures are marked with the symbol ↕.

the Manhattan distance to generate a well-dispersed set of supported points for MOCO problems. Such a set is defined as follows.

Definition 4.1 (Borges and Hansen (2002))**.** Let Λ^p be a compact subset of $\mathbb{R}^p$. A set $\Lambda \subset \Lambda^p$ is maximally dispersed for the Manhattan distance $d_1(\cdot,\cdot)$ w.r.t. Λ^p if for any $\Lambda' \subset \Lambda^p$ such that $|\Lambda'| > |\Lambda|$,

$$\min_{\substack{\lambda,\mu\in\Lambda'\\ \lambda\neq\mu}} d_1(\lambda,\mu) \leq \min_{\substack{\lambda,\mu\in\Lambda\\ \lambda\neq\mu}} d_1(\lambda,\mu)$$

where $d_1(\lambda,\mu) = \sum_{j=1}^{p} |\lambda_j - \mu_j|$ for all $\lambda, \mu \in \mathbb{R}^p$.

A maximally dispersed set w.r.t. $\Lambda^p = \{\lambda \in \mathbb{R}^p_{\geqq} : \sum_{j=1}^{p} \lambda_j = 1\}$ can be easily generated using the following result:

Proposition 4.2 (Borges and Hansen (2002))**.** *For any positive integer q, the set*

$$\Lambda = \left\{ \lambda \in \mathbb{R}^p_{\geqq} : \sum_{j=1}^{p} \lambda_j = 1, \lambda_j \in \{1/q, 2/q, \ldots, 1\}, j = 1, \ldots, p \right\}$$

is maximally dispersed w.r.t. $\{\lambda \in \mathbb{R}^p_{\geqq} : \sum_{j=1}^{p} \lambda_j = 1\}$ and $|\Lambda| = C^q_{p+q-1}$

We propose to use weight vectors obtained from the dichotomic scheme in the case $p = 2$, due to the efficiency of the procedure, and maximally dispersed weights in the case $p \geq 3$. In both cases optimal points for the corresponding weight vectors are computed and yield an update of the search region. This corresponds to Steps 2-5 in Algorithm 4.1. The procedure `updateSearchRegion(`z, $\updownarrow N$, $\updownarrow U(N)$`)`, the implementation of which is detailed in Section 4.4.4, inserts a point z in N if it is not dominated (according to "$\leqq$") and updates the upper bound set $U(N)$.

Additionally, we consider the computation of feasible points to be inserted in the initial set N just after Step 5 through a neighborhood search heuristic (see Section 4.4.2).

4.1.2 Association between weight vectors and search zones

In the exploration process of the search region, it is basically possible either (1) to iterate over local upper bounds of $U(N)$ or (2) to iterate over directions of Λ.

For (1), a local upper bound is selected at each iteration and a ranking direction is chosen and exploited to explore the corresponding search zone. In this case a ranking direction has to be considered several times.

Iterating over ranking directions (2) makes it possible to group local upper bounds together. In this case a group of local upper bounds is associated to the current ranking direction which is then exploited to explore all corresponding search zones. This is actually the way chosen by Przybylski et al. (2010b) for their multi-objective two-phase method. As

we will see, this makes it necessary to compute one optimum for each ranking direction in order to do the association. However, contrary to (1) we can define from the set of local upper bounds associated to the current ranking direction an upper bound on the weighted sum value of any point we want to rank along a given direction. Defining such a weighted sum upper bound often makes it possible to prune the ranking tree, which is especially the case for the Minimum Spanning Tree problem.

Therefore, we choose to follow (2), i.e. to group search zones together.

It remains to define a criterion for the association between ranking directions and search zones. In the multi-objective two-phase method of Przybylski et al. (2010b), a local upper bound u is associated to the ranking direction λ such that the Euclidean distance between u and the hyperplane h passing through $z^{*\lambda} = \arg\min\{\lambda \cdot z : z \in Y\}$ whose normal is λ, namely

$$d^*_{\mathrm{E}}(\lambda, u) = \frac{\lambda \cdot z^{*\lambda} - \lambda \cdot u}{\sqrt{\sum_{j=1}^{p} \lambda_j^2}}$$

is minimal. We also consider a simpler criterion, namely the minimization of the normalized difference

$$d^*_{\mathrm{D}}(\lambda, u) = \frac{\lambda \cdot z^{*\lambda} - \lambda \cdot u}{\sum_{j=1}^{p} \lambda_j}.$$

Preliminary experiments, which we do not report here, showed that these criteria behave similarly in the whole method. Therefore, we choose to rely in the remainder on the minimal normalized difference criterion.

The function `associateSearchZones(`$U(N)$`, `Λ'`, `λ`)` associates to the ranking direction $\lambda \in \Lambda'$ a subset of $U(N)$ according to one of the criteria described above, where Λ' is a set of ranking directions that have not been used for ranking points. If we consider the minimal normalized difference as association criterion, `associateSearchZones(`$U(N)$`, `Λ'`, `λ`)` returns

$$\left\{ u \in U(N) : \arg\min_{\lambda' \in \Lambda'} d^*_{\mathrm{D}}(\lambda', u) = \lambda \right\}.$$

Note that the association function assumes that weighted sum optima are known for each $\lambda \in \Lambda$. That is why they are computed at Step 4 of Algorithm 4.1.

4.1.3 Exploitation of a ranking direction to explore a group of search zones

Following the discussion in the above section, we assume that a ranking direction λ has been associated to a subset S_λ of search zones defined by a set U_λ of local upper bounds (i.e. $S_\lambda = \bigcup_{u \in U_\lambda} C(u)$).

Points are ranked along a ranking direction λ by calling `computeNextPoint(`λ`)`, which returns a point whose weighted sum value according to λ is optimal if we exclude the optimal point computed at Step 4 and the points obtained from earlier calls to this procedure. The

function `ubsTest`(λ, z, U_λ) compares the weighted sum value of a point z w.r.t. λ to the value of the local upper bounds of U_λ. It returns **true** if

$$\max_{u \in U_\lambda} \lambda \cdot u \leq \lambda \cdot z \tag{4.1}$$

in the general case ($Y \subset \mathbb{R}^p$) and

$$\max_{u \in U_\lambda} \lambda \cdot (u - \mathbf{1}) < \lambda \cdot z. \tag{4.2}$$

in the case $Y \subset \mathbb{Z}^p$. According respectively to Rule "$\mathbb{R}$" and Rule "$\mathbb{Z}$" from Section 3.1.5, if the test is satisfied, then z as well as all points of Y whose weighted sum value according to λ is greater than or equal to $\lambda \cdot z$ lie outside the partial search region S_λ. Practically, the left-hand side of (4.1) or (4.2) is computed only when U_λ is created or updated.

Now we consider the consequences induced by the discovery of a point z obtained while ranking along λ. We assume that we are within the general case of Rule "$\mathbb{R}$", and we denote by α the left-hand side in (4.1). We discuss four situations with respect to z, which are also illustrated in Figure 4.1.

(a) z belongs to a search zone of S_λ. Then U_λ is updated and α may be lowered.

(b) z belongs to a search zone of $S(N) \setminus S_\lambda$. Although it has no consequence on S_λ itself, $U(N)$ should nevertheless be updated, which reduces the future effort to explore the remaining zones.

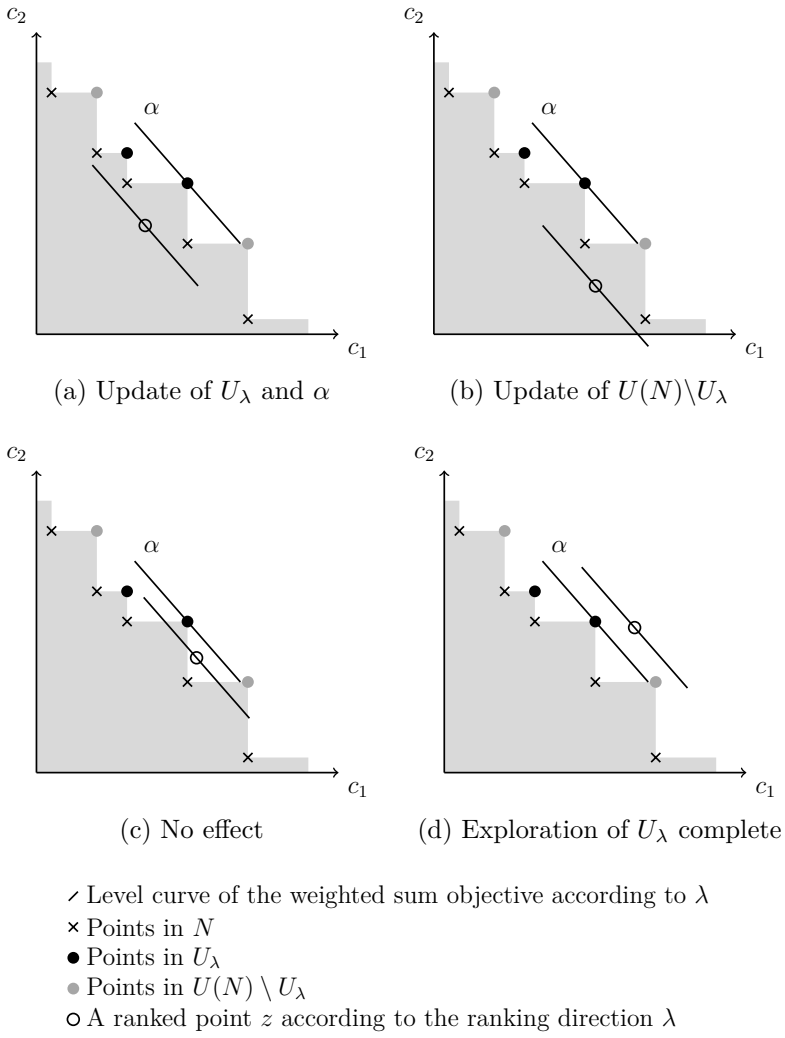
(c) z does not belong to any search zone of $S(N)$. Then z is dominated or already known and can be discarded.

(d) z is also dominated but its weighted sum value according to λ is greater than or equal to α. This guarantees that all feasible points of S_λ have been generated. In this case, the search zones of S_λ can be safely removed from $S(N)$.

These operations are performed at each iteration of the **while** loop (Step 7).

4.1.4 Organizing the whole search

The exploration of the whole search region is done considering ranking directions one by one, together with the associated search zones. It remains to define how the current weight vector is chosen at Step 8. This can be done according to the criterion that associates them to search zones. Przybylski et al. (2010b) propose to select, among the remaining ranking directions (i.e. of Λ'), the direction λ that minimizes the maximal value of the association criterion of `associateSearchZones` among the local upper bounds U_λ associated to λ. This corresponds to the idea of prioritizing the groups of search zones that are likely to be quickly explored.



(a) Update of U_λ and α

(b) Update of $U(N)\backslash U_\lambda$

(c) No effect

(d) Exploration of U_λ complete

Figure 4.1: Consequences linked to the computation of a new extension of a search node

This depends on the association criterion that was retained in `associateSearchZones` (Przybylski et al. (2010b) again considered the Euclidean distance) and assumes that the association operation is performed for each $\lambda \in \Lambda'$ at each loop of Step 7. Moreover, since the ranked points induce updates of $U(N)$, the order on the set of all ranking directions cannot be determined at the beginning of the procedure. Namely, in the case of the minimal normalized difference criterion, the current weight vector is selected as a solution to

$$\min_{\lambda \in \Lambda'} \max_{u \in U_\lambda} d^*_{\mathrm{D}}(\lambda, u)$$

where $\{U_\lambda\}_{\lambda \in \Lambda'}$ is a partition of $U(N)$ according to `associateSearchZones`, i.e. $U_\lambda = \{u \in U(N) : \arg\min_{\lambda' \in \Lambda'} d^*_{\mathrm{D}}(\lambda', u) = \lambda\}$, for all $\lambda \in \Lambda$.

4.2 Integration of the ranking strategy in a branching tree

We presented in the previous section a ranking strategy to generate the nondominated set. This ranking strategy is integrated into the bounding procedure of a branch and bound algorithm, therefore it operates on a search node rather than on the whole instance.

The hybrid strategy is based on two procedures: the branching procedure, which is the main procedure (Algorithm 4.2 – Section 4.2.1) and the bounding procedure `exploreBB` (Algorithm 4.3 – Section 4.2.2). Two functions which characterize the hybrid approach are added, namely `stoppingCriterion` (Section 4.2.3) and `chooseBranchingItem` (Section 4.2.4).

4.2.1 Branching scheme

In Algorithm 4.2, a search node s is defined by a set of included or mandatory items and a set of excluded or forbidden items respectively denoted by $\mathrm{in}(s)$ and $\mathrm{out}(s)$, as well as a *proper* search region represented by an upper bound set $U(s)$. All this information is kept together with s and is hence passed to subroutines that operate on s.

We consider in the description of the approach as well as in our implementation a binary branching scheme, which is defined by the choice of a *branching item* which is made mandatory in one child and forbidden in the other child.

The role of the procedure `exploreSearchNode` is to perform some computations on a search node s before a possible branching in order to determine if this node s can be pruned.

Algorithm 4.2: Main hybrid procedure

input : instance data, $\boldsymbol{M}$ – *$\boldsymbol{M}$ is strictly dominated by any feasible point*
output : N – *the nondominated set*

1 $\Lambda \leftarrow$ `computeWeightVectorSet()` – *Computation of a set of weight vectors*
2 $N \leftarrow \emptyset$ – *Initialization of the output*
– *Initialization of the root node r and the search list O*
3 $\text{in}(r) \leftarrow \emptyset$; $\text{out}(r) \leftarrow \emptyset$; $U(r) \leftarrow \{\boldsymbol{M}\}$
4 $O \leftarrow \{r\}$ – *List of open nodes*
– *Main search loop*
5 **while** $O \neq \emptyset$ **do**
6 Select $s \in O$; $O \leftarrow O \setminus \{s\}$ – *Choice of a search node to explore*
7 dominated $\leftarrow$ `exploreSearchNode(`$\updownarrow s$, $\updownarrow N$, Λ`)`
8 **if not** dominated **then**
9 Select a branching item e
– *Generate children of s*
10 $\text{in}(s_e) \leftarrow \text{in}(s) \cup \{e\}$; $\text{out}(s_e) \leftarrow \text{out}(s)$; $U(s_e) \leftarrow U(s)$
11 $\text{in}(s_{\bar{e}}) \leftarrow \text{in}(s)$; $\text{out}(s_{\bar{e}}) \leftarrow \text{out}(s) \cup \{e\}$; $U(s_{\bar{e}}) \leftarrow U(s)$
12 $O \leftarrow O \cup \{s_e, s_{\bar{e}}\}$
13 **return** N

4.2.2 Bounding procedure

The bounding procedure, `exploreSearchNode`, is described in Algorithm 4.4. It is based on Algorithm 4.1 but it first requires some modifications due to the fact that it is applied on a search node s.

First, we replace Steps 3-5 in Algorithm 4.1, which compute weighted sum optima for all weight vectors of Λ, by a dedicated procedure, Algorithm 4.3. This procedure avoids the computation of all weighted sum optima w.r.t. Λ if a subset of Λ is sufficient to prove that $Y(s) \cap S(N) = \emptyset$. In this case we get $U(s) = \emptyset$, therefore Algorithm 4.4 stops and the search node s is pruned. Otherwise, all weighted sum optima are computed and the remaining local upper bounds are associated to the weight vectors of Λ as in Algorithm 4.1.

The search zones associated to local upper bounds that are removed from $U(s)$ at Step 8 of Algorithm 4.3 and at Step 11 of Algorithm 4.4 are guaranteed to contain no point of $Y(s)$. This will also be the case for any descendant of s, which justifies that only the local upper bound in $U(s)$ left active by `exploreSearchNode` need to be passed to its children nodes. This is actually what is done in the main procedure (Algorithm 4.2).

If at the end of Algorithm 4.4 we have $U(s) \neq \emptyset$, we propose to use the association criterion again in order to (1) define an order on the weight vectors of Λ to be used in Algorithm 4.3 for the children nodes of s and (2) define an order on the list O of open search nodes in Algorithm 4.2.

Algorithm 4.3: Procedure `firstStage`

in/out : s, N – *search node, current set of points*
input : Λ – *set of weight vectors*

```
1  O ← U(s)                                   – List of search zones to explore
2  while O ≠ ∅ do
3      Select a local upper bound u ∈ O ; O ← O \ {u}
4      for λ ∈ Λ do
5          Let z^{*λ}(s) = arg min{λ · z : z ∈ Y(s)}
6          updateSearchRegion(z^{*λ}(s), ↕ N, ↕ U(s))
7          if ubsTest(λ, z^{*λ}(s), {u}) then
               – Remove u from U(s) since its associated search zone
               – does not intersect Y(s)
8              U(s) ← U(s) \ {u}
9              break                          – Skip to the next element of O
```

Algorithm 4.4: Procedure `exploreSearchNode`

in/out : s, N – *search node, current set of points*
input : Λ – *set of weight vectors*

```
   – Bounding part based on weighted sum optima
1  firstStage(↕ s, ↕ N, Λ)
2  Λ' ← Λ
3  while U(s) ≠ ∅ and Λ' ≠ ∅ do
4      Select λ ∈ Λ'; Λ' ← Λ'\{λ}
       – Association of search zones from U(s) to a weight vector λ
5      U_λ ← associateSearchZones(s, U(s), Λ', λ)
6      if U_λ ≠ ∅ then
           – Perform the computation of ranked points
7          while U_λ ≠ ∅ and not stoppingCriterion(s, λ) do
8              z ← computeNextPoint(↕ s, λ)
9              updateSearchRegion(z, ↕ N, ↕ U(s))
10             if ubsTest(λ, z, U_λ) then
11                 U(s) ← U(s) \ U_λ
12                 U_λ ← ∅
13     if U(s) ≠ ∅ then
           – The exploration was interrupted due to stoppingCriterion.
14         br(s) ← chooseBranchingItem()
15         return false
16 return true
```

To this end we partition $U(s)$ according to Λ into $\{U_\lambda(s)\}_{\lambda\in\Lambda}$ with $U_\lambda(s) = \{u \in U(s) : \arg\min_{\lambda'\in\Lambda'} d_{\mathrm{D}}^{\mathrm{k}}(s,\lambda',u) = \lambda\}$, for all $\lambda \in \Lambda$ where k indicates that the association criterion is computed with respect to the last ranked point of $Y(s)$ along λ. For (1), the weight vectors of Λ are sorted in increasing order of $\max_{u\in U_\lambda(s)} d_{\mathrm{D}}^{\mathrm{k}}(s,\lambda,u)$. For (2), the search nodes are sorted in increasing order of $\max_{\lambda\in\Lambda}\max_{u\in U_\lambda(s)} d_{\mathrm{D}}^{\mathrm{k}}(s,\lambda,u)$, giving higher priority to the search nodes that are expected to be quickly completely explored.

4.2.3 Stopping criterion

The exploitation of a ranking direction may be at some point inefficient, in the sense that it yields only points that lie outside the search region. The function `stoppingCriterion` aims at stopping the exploitation of a ranking direction (in which case it returns **true**) according to the information gathered on the current search node.

We propose to exploit a ranking direction as soon as a maximal number of successively computed points that lie outside the search region is not reached. If the stopping criterion is met, then the exploration continues with the next ranking direction.

Several values for this criterion are considered in the computational experiments.

4.2.4 Choice of a branching item

The role of procedure `chooseBranchingItem` is to optimize the result of the branching step. If Algorithm 4.4 fails to explore a search node s, i.e. to prove that $Y(s) \cap S(N) = \emptyset$, we would like to share the difficulties encountered by the ranking algorithm between the children nodes of s. To this end, we propose to exploit the information computed through the application of the ranking algorithm. In particular, we are interested in the weight vector such that the associated instance of the ranking algorithm is the farthest from its goal to explore its associated search zones.

Consider the above partition of $U(s)$ into subsets $U_\lambda(s)$, $\lambda \in \Lambda$, according to $d_{\mathrm{D}}^{\mathrm{k}}(s,\lambda,u)$ and a weight vector λ_{br} which maximizes $\max_{u\in U_\lambda(s)} d_{\mathrm{D}}^{\mathrm{k}}(s,\lambda,u)$. Then, as a heuristic to balance the computational effort to explore the children of s, we choose a branching item in $E \setminus (\mathrm{in}(s) \cup \mathrm{out}(s))$ that best splits the solutions that were ranked along λ_{br}. Namely, let R be the set of solutions of $X(s)$ that were ranked along λ_{br}. We choose as branching item any $e \in E$ such that the number of solutions of R that take e best fits half of the total number of solutions of R.

Note that in the extreme case of a branch and bound strategy without ranking the above criterion to select a branching edge is meaningless. In this case, we propose instead to branch first on items whose outcomes are the most unbalanced. To this end, we evaluate each item $e \in E$ according to

$$\max_{j_1,j_2\in\{1,\ldots,p\}} c_{j_1}(e) - c_{j_2}(e)$$

and then we select the item of $E \setminus (\text{in}(s) \cup \text{out}(s))$ which maximizes this quantity.

4.3 Computation of a reduced $(1+\varepsilon)$-approximation

The method we propose can be easily adapted to generate a reduced $(1+\varepsilon)$-approximation X_ε or, equivalently, a reduced approximate nondominated set Y_ε. As in the "exact" version, we will avoid referring to the decision space when unnecessary.

A tolerance $\varepsilon > 0$ may be introduced in the weighted sum test in `ubsTest` (Section 4.3.1) and also in the update of the current set of points N and of the search region done by `updateSearchRegion` (Section 4.3.2). We will consider without loss of generality the root search region $S(N)$.

4.3.1 Tolerance in the weighted sum test

The weighted sum tests in Section 4.1.3 and Section 4.2.2 rely on Rule "$\mathbb{R}$" or Rule "$\mathbb{Z}$", depending on the hypothesis on the objective space, which were presented in Section 3.1.5. The tolerance can be simply introduced in the test by substituting any point z by $(1+\varepsilon)z$.

The introduction of the tolerance ε can also be interpreted as shifting the local upper bounds that define the region by a factor of $\frac{1}{1+\varepsilon}$ (we call this ε-shift of the search region for short). A simple situation is illustrated in Figure 4.2a.

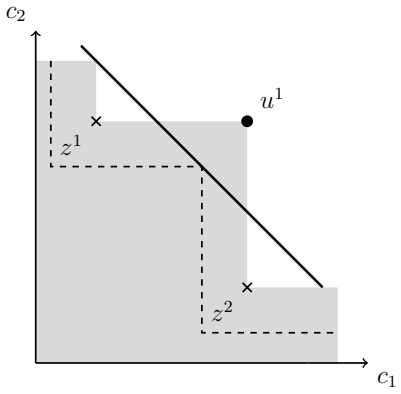
Therefore it is not hard to see how `ubsTest` can be adapted. For example, the test (4.1) to determine if a set of search zones associated to an upper bound set U_λ contains non dominated point is modified as follows:

$$\max_{u \in U_\lambda} \lambda \cdot u < \lambda \cdot (1+\varepsilon)z$$

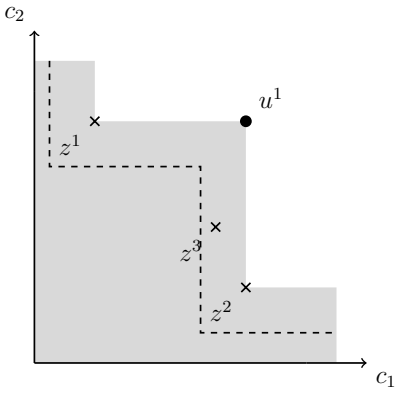
4.3.2 Case of ε-dominated points

Since we are generating reduced sets, dominated points will not be kept. Among points that are currently non dominated, i.e. which lie in the current search region, some may be ε-dominated, such as the point z^3 in Figure 4.2b which is ε-dominated by z^2 (we assume that $z^1, z^2 \in N$).

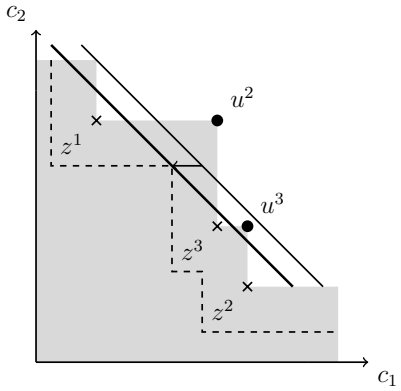
ε-dominated points may be discarded since there will be some points in the output that cover them (A). But it is also possible to include them in N and update the search region accordingly (B). The main advantage in strategy (B) is that some search zones will be refined and weighted sum upper bounds computed from local upper bounds may decrease. This is illustrated in Figure 4.2c where the search region is updated with z^3, which yields two new local upper bounds, u^2 and u^3. The weighted sum upper bound formerly computed from u^1 decreases.



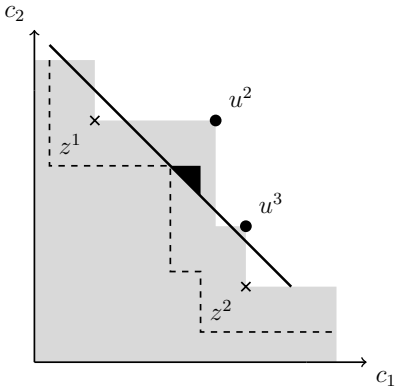
(a) ε-shifted search region and level curve $\lambda \cdot z = \lambda \cdot \frac{u}{1+\varepsilon}$ (in bold)



(b) ε-dominated point z^3 is discarded.



(c) ε-dominated point z^3 is included in N and in the search region.



(d) ε-dominated point z^3 is not inserted in N but the search region is updated according to z^3. Potentially missed points lie in the black triangle.

Figure 4.2: Example situations for the approximation version (log scales). Dashed lines indicate the boundary of the ε-shifted search region.

Note that if the search region is updated with ε-dominated points, then they must be included in N as well. Otherwise, some points may be eventually non covered. Figure 4.2d shows a black triangle which may contain uncovered points if the search region is updated according to z^3 but z^3 is not included in N.

So in strategy (A), the dominance relation in Step 3 of `updateSearchRegion` is replaced by the ε-dominance relation $\leqq_\varepsilon$. In strategy (B), `updateSearchRegion` is not modified. It can be expected that strategy (A) generates fewer points than strategy (B) but the latter may be more time efficient than the former, due to the increased opportunities to reduce weighted sum upper bounds. This will be investigated in our experiments.

4.4 Instantiation on the Minimum Spanning Tree problem and implementation issues

The first three subsections (Section 4.4.1, Section 4.4.2, and Section 4.4.3) concern the instantiation of the proposed approach on the MOST problem. The last subsection (Section 4.4.4) deals with the implementation of the procedure `updateSearchRegion`.

4.4.1 Ranking spanning trees

In order to obtain solutions to the single-objective Minimum Spanning Tree problem in non decreasing order, we implemented the algorithm of Katoh et al. (1981) presented in Section 2.5.

For the initialization of the algorithm, we propose to use the procedure based on Kruskal's algorithm to compute a minimum spanning tree and a spanning tree verification algorithm to generate the associated edge exchanges (see Section 2.5.2). This procedure seems more appealing than the procedure based on Prim's algorithm. Its time complexity reduces to $O(m\alpha(m,n)$, where $\alpha(m,n)$ is the inverse Ackermann function (see e.g. Tarjan (1983)) – a very slowly growing function) if the edges are presorted. Moreover, it distinguishes the computation of a minimum weight spanning tree and the computation of the associated edge exchanges, which makes it possible to do the latter only at the required time.

Use of an upper bound From the description of the procedure, we see that if no spanning tree of cost more than a specified α is needed, we can easily reduce the computational effort and the required space. For any enumerated spanning tree t, we discard t-exchanges (e,f) such that $w(t)+w(e,f)>\alpha$.

4.4.2 Computation of an initial solution pool through neighborhood search

We stressed that non supported points may be introduced at the initialization of the ranking or hybrid approaches in order to refine the initial search region. Such points can be obtained using a *neighborhood* or *local* search heuristic. Neighborhood search in the case of the Minimum Spanning Tree problem often yields a reduced efficient set. For this reason, it was used in several works (see e.g. Andersen et al., 1996 and Sourd and Spanjaard, 2008). We propose an efficient implementation of neighborhood search for the MOST problem, which we describe in Algorithm 4.5.

First, the computation of all cycles required by Step 4 can be done in $O(m\alpha(m,n))$ time using the least common ancestors algorithm of Tarjan (1979) (see also Cormen et al., 2009). It could also be done in linear time using the algorithm of Gabow and Tarjan (1985), however we believe that it would not be significant on the practical efficiency.

Second, at Step 9, we apply the move-to-front heuristic of Bentley et al. (1993) to lists P and O. This heuristic concerns linear lists of solutions or points. If an element of such a list dominates a candidate element to be inserted, then, while the latter is not inserted, the former is moved to the front of the list. The moved element, which is expected to dominated some future candidates, will then be quickly accessed. We refer to Bentley et al. (1993) for the theoretical foundations of this procedure.

It is required that Algorithm 4.5 starts from at least one efficient spanning tree in P. It can be for example any supported spanning tree. Although the adjacency subgraph of efficient spanning trees is not connected in general (Ehrgott and Klamroth, 1997), the adjacency subgraph of supported spanning trees is (see e.g. Andersen et al., 1996 or Okamoto and Uno, 2011). So the neighborhood search algorithm described above will return all supported spanning trees, provided it is initialized with at least one supported spanning tree. The expected benefit of initializing Algorithm 4.5 with many or even all supported points is that dominated neighbors will be discarded earlier. Our computational experiments, however, showed that it is not always relevant in practice, and the largest time save we observed on the procedure was about 25%.

4.4.3 Preprocessing search nodes

We presented in Section 2.3.1 preprocessing rules proposed by Sourd and Spanjaard (2008) that extend the well known coloring rules for the Minimum Spanning Tree problem. These rules can be applied for the whole instance, i.e. at the root of the search tree, but it is also possible to apply the preprocessing rules at each search node.

We considered a binary branching scheme that partitions a search node s into two subnodes according to a branching edge e, namely s_e for which e is made mandatory and $s_{\bar{e}}$ for which e is made forbidden. Edge e can be equivalently viewed as a *blue* edge in s_e and as

Algorithm 4.5: Implementation of a neighborhood search procedure for the MOST problem

input : $G = (V, E)$, c, P – *Graph, vector criterion function, initial list of spanning trees*
output : P

```
1  O ← P                                              – List of spanning trees to visit
2  while O ≠ ∅ do
3      Select t ∈ O ; O ← O \ {t}                     – Take the head of O
4      for f ∈ E \ t, e ∈ Cycle(t, f) do
5          t' ← t ∪ f \ e
6          dominated ← false
7          for t'' ∈ P do
8              if c(t'') ≦ c(t') then
9                  Move t'' to the front of P and O (if applicable)
10                 dominated ← true
11                 break
12             else if c(t') ≤ c(t'') then
13                 P ← P \ {t''}
14                 O ← O \ {t''} (if applicable)
15         if not dominated then
16             Insert t' in P and O
17     return P
```

a *red* edge in $s_{\bar{e}}$, which is an arbitrary coloring stemming from the branching scheme. Now, coloring e blue will not make it possible to color additional edges in blue but it may enable to color more red edges, since it will reduce the set of uncolored edges in a fundamental cycle (containing no red edge) that must dominate a candidate red edge. In the same way, only the blue rule may apply after coloring e red.

We summarize this in the following proposition.

Proposition 4.3. *Consider two subnodes s_e and $s_{\bar{e}}$ of s that respectively make edge e mandatory and forbidden. Then, if s was preprocessed through the blue and red preprocessing rules, only the red rule may apply to s_e and only the blue rule may apply to $s_{\bar{e}}$.*

4.4.4 Update of the search region

The role of procedure `updateSearchRegion`, described in Algorithm 4.6, is twofold:

1. determine if a candidate point z lies in the search region defined by an upper bound set $U(N)$,
2. if the answer to the first point is "yes", perform the update of N and $U(N)$.

Algorithm 4.6: Procedure `updateSearchRegion`

```
input  : z                                  – candidate point
in/out : N, U(N)                            – set of points, set of local upper bounds
1  updated ← true
2  for z' ∈ N do
3      if z' ≦ z then
4          Move z' to the front of N
5          updated ← false
6          break
7      else if z ≤ z' then
8          N ← N \ {z'}
9  if updated then
10     Update U(N) with respect to z
11 return updated
```

Testing membership to the search region

The first point can be achieved either (a) by comparing the candidate z to the points of N or (b) by comparing it to the local upper bounds of $U(N)$. Depending on z, there are two situations:

- If z is not in the search region, this will be proved using (a) by finding a point of N which dominates z and no update of the search region is required. Using (b) would require either to scan all local upper bounds of $U(N)$, or to use some specialized data structure, both to obtain that $z \not< u$ for all $u \in U(N)$. From Section 3.4.1, $U(N)$ can be far larger than N, especially for $p \geq 4$.

- If z is in the search region, this can be proved using (a), which requires to scan all points of N, or using (b), which requires to find a local upper bound u such that $z < u$. However, the update of the search region (point 2) requires that all local upper bounds satisfying $z < u$ be found.

So the privileged method among (a)-(b) to achieve point 1 depends on whether candidate points are likely to lie in the search region or not. Due to the fact that the points are obtained using a ranking procedure, which generates many equivalent or dominated points, we retain method (a). We use again the move-to-front heuristic of Bentley et al. (1993) presented in Section 4.4.2 to perform insertions in the set N, implemented as a linear list.

Updating upper bound sets

Chapter 3 provides several algorithms to update an upper bound set. Here we cannot assume that the points of Y are in "general position" (in the sense that on any objective function, no two points of Y share the same component value) especially because they correspond to sums of objective values. Therefore, we should use Algorithm 3.3 or Algorithm 3.5. According to our computational experiments in Section 3.4.3, Algorithm 3.3 should be preferred for $p \leq 4$ while Algorithm 3.5 is more efficient for $p \geq 5$.

Special care should be taken due to the fact that several upper bound sets exist at the same time. Namely, a local upper bound which is updated in $U(N)$ should also be updated in all other upper bound sets. To this end, u is stored only once and keeps pointers to the upper bound sets that contain u.

4.5 Computational experiments

We implemented the proposed approach in C and ran it on several instances on a workstation equipped with an Intel Core i7-3840QM CPU at 2.80GHz with 8MB cache and 32GB RAM. We first describe the instance types, then we detail the algorithm parameters and computed results, and finally we present the experiments and their results. Our approach is compared in the bi-objective case to the approach of Sourd and Spanjaard (2008).

4.5.1 Experimental setup

Instances

Our test instances are defined through the following parameters:

- the graph type t: we consider cliques (c), connected graphs whose edges are created with fixed probability π ($d\pi$), square grids (g), and cliques with an additional gadget which makes it possible to duplicate the Pareto front (m);
- the type of random generation of the edge objective values r: we consider uniform edge costs (u) and conflicting edge costs (c);
- the number of vertices n;
- the number of objective functions p;
- the objective range K.

For convenience, test instances will be denoted by a string $t/r/n/p/K$ where $t \in \{c, g, m, d\pi\}$, $r \in \{u, c\}$ and n, p, K are positive integers. Now we detail the graph and objective values generators.

Graph types Cliques correspond to complete graphs. Grids have $a^2 = n$ vertices arranged as a $a \times a$ grid where horizontally and vertically neighboring vertices are joined together.

Multiple Pareto front graphs are cliques with an additional gadget consisting of edges $(n, n+1), (n, n+2), \ldots, (n, n+p)$ with respective criterion vectors

$$(M, 0, \ldots, 0), (0, M, 0, \ldots, 0), \ldots, (0, \ldots, 0, M)$$

and edges $(n+1, n+2), \ldots, (n+p-1, n+p)$ all of null cost. Since all null cost edges will be included in any efficient solution, only one of the first p gadget edges will be included. The parameter M is chosen as a sufficiently large integer such that the nondominated points of the sub-instances respectively including the edges of cost $(M, 0, \ldots, 0), \ldots, (0, \ldots, 0, M)$ are also nondominated points of the whole instance. This device was proposed by Knowles and Corne (2001) and used in Sourd and Spanjaard (2008).

Random objective values types Uniform objective values are randomly chosen in $\{0, \ldots, K\}$ in a uniform way.

Conflicting objective values are used to design instances whose edges (1) are unlikely to dominate each other and (2) whose outcomes are highly conflicting. We describe a way to obtain such instances in the bi-objective case. For a each edge e and for $c_1(e)$ we select

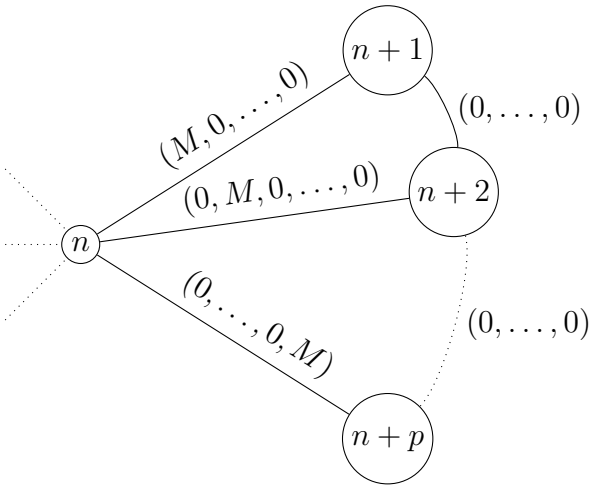


Figure 4.3: An instance gadget

with probability 1/2 an interval I_1 corresponding to the first half or to the second half of $\{0,\ldots,K\}$ and we define I_2 as the other half of 1/2. Then the idea is to draw $c_j(e)$ in the interval I_j, for $j=1,2$. To satisfy both (1) and (2) we propose to draw in I_j according to an approximately Gaussian distribution centered at the middle of I_j. Since we want $c_j(e) \in I_j$, we choose a Binomial distribution with probability 1/2 and 100 trials. Figure 4.4 shows the density of such a distribution for $c_1(e)$.

Parameters combinations We consider the following parameter combinations: $c/u/n/p/K$, $c/c/n/p/K$, $m/u/n/p/K$, $d\pi/u/p/n/K$, and $g/u/n/p/K$. We represent in Figure 4.5 density plots of the set Y of all feasible points for some representative instances from these parameter combinations. Two aspects are important to observe on these plots: the shape of the Pareto front, and in particular the location of non supported nondominated points and the density of Y around the Pareto front. Indeed these two aspects play an important role in the behaviour of the search strategies we test below.

4.5.2 Results

We summarize and comment below our experimental results.

- All experiments where performed on 10 random instances of each type.
- Unless specified, the values are the averages of the values obtained on a set of identical types of instances.
- All times are CPU times measured in *seconds*.
- The memory consumption is measured in MB.

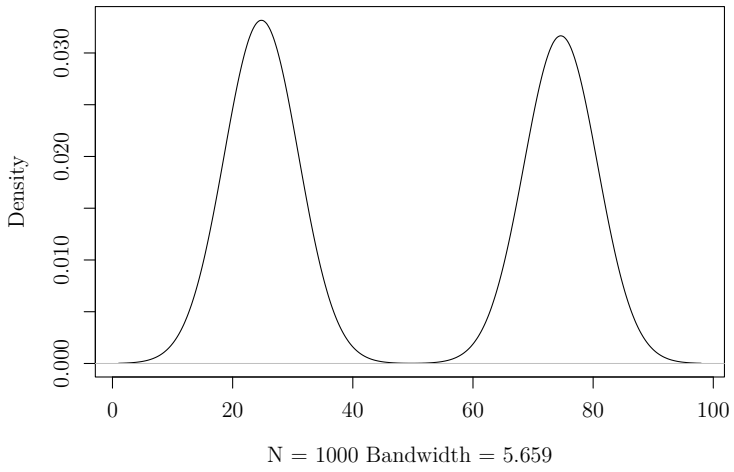


Figure 4.4: Density of the distribution of c_1 for conflicting instances

General statistics on instances

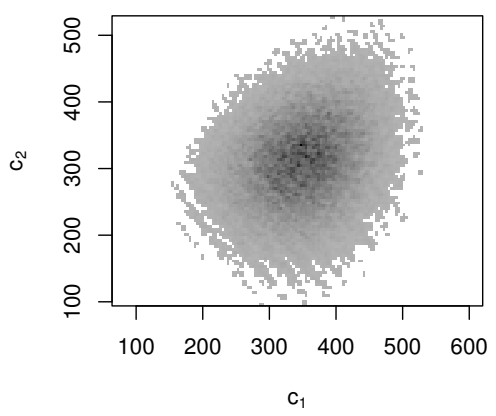
We provide in Table 4.1 the list of our test instances with the number of nondominated points and the proportion of extreme nondominated points. We show how many edges can be colored after the application of the preprocessing rules presented in Section 2.3.1. on the whole instance. The column "Chosen Λ" indicates the method chosen in the remainder for obtaining a set of weight vectors. "D" refers to the bi-objective dichotomic scheme and numbers refer to the choice of the parameter q in the determination of a maximally dispersed set. This number was chosen according to preliminary experiments.

Comparison of strategies

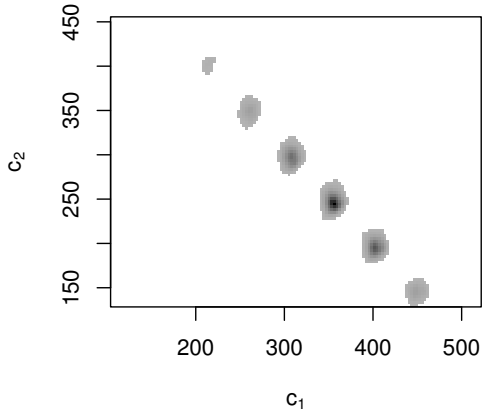
Tables 4.2 and 4.3 provide respectively computation times and memory consumptions of four strategies. All these strategies rely on the minimal normalized difference association criterion. Strategy S1 is ranking only, strategy S4 is branch and bound only, and strategies S2 and S3 are hybrid strategies for which the values of the stopping criterion are respectively 1000 and 100. For S1, the set of points N is fed with neighborhood search after the computation of weighted sum optima at the root node. The symbol "—" indicates instances that could not be solved using the corresponding strategy, either because the computation time was too long ($>$ 1 hour) or because the available memory was exceeded.

The following observations and analysis can be made on these results.

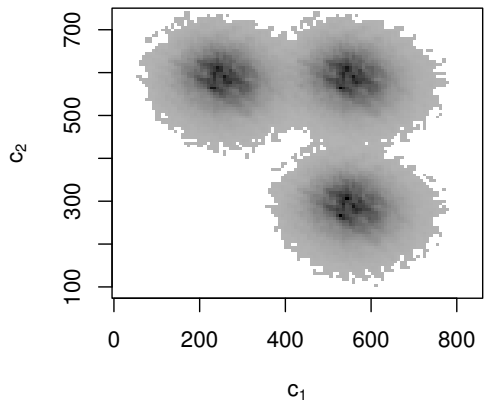
- The ranking strategy S1 is time-efficient on most instances, except:



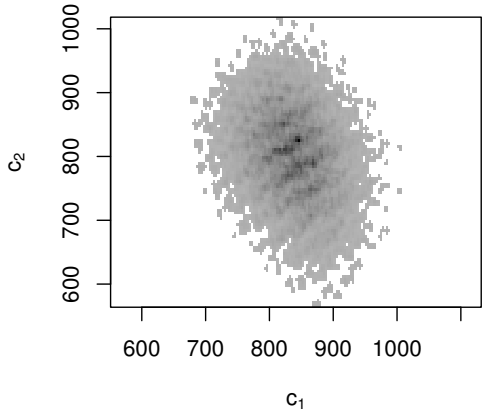
(a) $c/u/7/2/100$



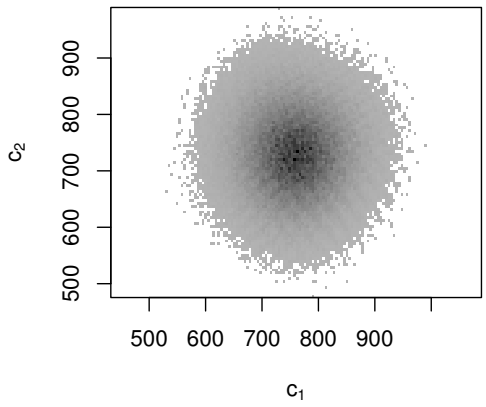
(b) $c/c/7/2/100$



(c) $m/u/7/2/100$



(d) $d0.1/u/20/2/100$



(e) $g/u/16/2/100$

Figure 4.5: Density plots of some representative instances

							Y_{nd}			$Y_{\text{extr}}/Y_{\text{nd}}$ (%)			
t	r	p	n	K	red	blue	min	mean	max	min	mean	max	Chosen Λ
c	u	2	100	100	4511.0	2.2	996	1164	1277	15.2	16.7	18.4	D
				1000	4492.0	1.1	3800	4933	6124	4.8	5.4	6.4	D
			200	100	19011.3	4.0	1610	1712	1801	15.2	16.5	17.2	D
				1000	18816.8	1.4	11253	12568	14042	4.5	4.9	5.3	D
			400	100	78132.7	7.9	2330	2460	2602	10.6	11.3	12.1	D
				1000	77387.0	1.9	23330	24476	26343	5.2	5.4	5.7	D
		3	15	10	51.4	0.5	446	711	1226	19.2	26.6	32.7	10
				100	40.0	0.0	1797	2844	4805	6.6	10.0	12.9	10
				1000	38.9	0.3	1666	3555	5844	5.8	8.1	12.7	10
			20	10	104.7	0.9	927	1441	2124	18.1	23.2	28.8	20
				100	84.3	0.0	8349	11195	20442	3.7	5.4	6.7	20
				1000	80.4	0.0	8026	15009	23621	3.2	4.4	6.2	20
		4	10	100	7.4	0.0	865	3047	7822	9.8	18.6	25.7	10
			12	100	8.8	0.0	5694	17195	40871	4.6	8.1	13.0	20
	c	2	10	100	16.2	0.0	68	143	181	8.6	11.1	19.1	D
			12	100	27.1	0.0	185	246	322	5.9	7.6	9.7	D
m	u	3	15	100	36.5	2.1	6486	9430	14913	2.5	3.5	4.4	20
$d0.1$	u	2	200	100	1356.8	16.5	3219	3768	4099	6.8	7.3	8.2	D
		3	30	100	2.4	7.3	352	1531	6764	5.3	15.0	23.4	20
g	u	2	100	100	28.4	42.0	406	522	682	8.9	10.5	12.1	D
			225	100	64.6	84.7	1617	2019	2418	5.1	6.3	7.5	D
			400	100	121.9	152.6	3719	4346	5694	4.1	5.1	5.8	D

Table 4.1: General information on the test instances

t	r	p	n	K	S1: Ranking	S2: Hybrid 1000	S3: Hybrid 100	S4: BB w. NS	SouSpa08
c	u	2	100	100	0.272	0.270	2.459	41.952	6.508
				1000	1.625	1.593	70.146	1258.932	129.549
			200	100	5.622	—	—	62.138	22.148
				1000	10.432	—	—	—	—
			400	100	—	—	—	13.562	154.559
				1000	56.463	—	—	—	—
		3	15	10	0.093	0.098	0.558	0.889	—
				100	0.575	2.532	5.554	10.529	—
				1000	0.883	4.542	8.517	15.734	—
			20	10	0.363	0.777	2.433	6.325	—
				100	5.759	45.503	118.944	253.876	—
				1000	11.087	96.913	229.357	419.488	—
		4	10	100	16.768	215.934	187.441	30.493	—
			12	100	280.399	—	—	2787.256	—
	c	2	10	100	41.018	31.958	18.882	4.920	2.397
			12	100	155.810	876.783	642.591	199.365	81.761
m	u	3	15	100	—	13.439	23.983	121.163	—
$d0.1$	u	2	200	100	1.697	2.900	56.447	1041.414	—
		3	30	100	0.493	—	—	—	—
g	u	2	100	100	0.071	0.071	0.237	4.670	1.181
			225	100	0.756	0.693	8.323	193.449	38.247
			400	100	3.878	5.949	87.230	1793.476	312.616

Table 4.2: Overview of strategy results
Average computation times (s)

t	r	p	n	K	S1: Ranking	S2: Hybrid 1000	S3: Hybrid 100	S4: BB w. NS
c	u	2	100	100	17.9	17.9	11.5	5.8
				1000	31.7	31.7	28.4	16.9
			200	100	2160.3	—	—	15.0
				1000	189.2	—	—	—
			400	100	—	—	—	36.0
				1000	472.5	—	—	—
		3	15	10	12.6	9.1	3.5	1.7
				100	43.8	30.0	11.1	5.7
				1000	43.3	33.6	12.4	7.9
			20	10	77.0	21.8	5.8	3.7
				100	79.7	75.8	33.3	30.0
				1000	140.1	101.1	45.3	47.2
		4	10	100	742.1	19.5	12.7	24.6
			12	100	898.5	—	—	555.2
	c	2	10	100	1811.0	13.9	2.1	1.2
			12	100	9935.2	21.2	2.7	3.7
m	u	3	15	100	—	216.9	31.1	25.4
$d0.1$	u	2	200	100	59.3	85.1	42.3	18.2
		3	30	100	13.4	—	—	—
g	u	2	100	100	6.1	5.8	5.2	2.5
			225	100	28.1	24.4	25.5	10.3
			400	100	113.4	115.9	80.0	30.6

Table 4.3: Overview of strategy results
Average memory consumption (MB)

  - on instances where the set of feasible points is dense around the Pareto front due to a low objective range compared to the solution sizes (low K/n), e.g. $c/u/2/400/100$,
  - on certain highly conflicting instances such as $c/c/10/2/100$ or $m/r/15/3/100$, which contain nondominated points which are far away from the boundary of the convex hull of the feasible points (bd(conv$Y$)).

  The strategy S1 however is the most memory-consuming. This comes from the requirement to keep nearly all ranked solutions for one ranking direction.

- The branch and bound strategy S4 is particularly sensitive to n and K: instances for which the product nK is high are hard to solve using this strategy. This includes instances for which the number of nondominated points is high but also instances made of sparse graphs (e.g. $t = g$ and $t = d0.1$) which contain nondominated points far away from bd(conv(Y)). These instances, however, differ from multiple Pareto front instances in the sense that they cannot be quickly partitioned into easy subinstances.

  The chosen order for search nodes makes this strategy cheaper on memory than the ranking strategy.

- The time performance of the hybrid strategies S2 and S3 lies generally between S1 and S4, except for multiple Pareto front instances and to a lesser extent for certain grid instances. For multiple Pareto front instances, hybrid strategies make good choices on the branching edges in the sense that the resulting subinstances are easier for the ranking strategy.

  Thanks to the stopping criterion, these hybrid strategies are often more efficient than the ranking strategy on memory consumption. For memory consumption again, we even see that the strategy S3 is the most efficient on a few instances ($c/u/20/3/1000$ and $c/c/12/2/100$). The memory requirement of a hybrid strategy (including ranking only and BB only) increases in the maximal size of the search tree and in the size of the largest ranking tree. Therefore strategy S3 actually achieves a compromise between the two which is better than other tested strategies on these instances.

Table 4.2 additionally shows computation times for the branch and bound algorithm of Sourd and Spanjaard (2008) on the bi-objective instances that could be solved. Their approach is more efficient than the branch and bound strategy S4 since it computes a weight vector set at each search node s based on conv$(Y)_{\geqq}$ and it maintains all the edge orderings induced by all possible weighted sum objectives (see Section 2.4.2). These two features are, to some extent, specific to the bi-objective case. The poorer performance of the approach of Sourd and Spanjaard on $c/u/400/2/1000$ is due to the time spent on neighborhood search (131.38 s, i.e. 85 % of the total time).

			S1: Ranking		S2: Hybrid 1000		S3: Hybrid 100		S4: BB		
p	n	K	without NS	with NS	without NS	with NS	without NS	with NS	without NS	with NS	NS time
2	200	100	5.622	1.741	—	—	—	—	—	62.138	0.471
		1000	10.432	21.973	—	—	—	—	—	—	6.910
	400	100	—	9.215	—	—	—	—	—	13.562	1.710
		1000	56.463	130.459	—	—	—	—	—	—	42.337
3	15	10	0.093	0.076	0.098	0.214	0.558	0.637	4.407	0.889	0.025
		100	0.575	0.902	2.532	4.260	5.554	7.288	47.968	10.529	0.312
		1000	0.883	1.423	4.542	6.724	8.517	11.171	64.465	15.734	0.539
	20	10	0.363	0.272	0.777	0.591	2.433	2.153	39.039	6.325	0.074
		100	5.759	12.452	45.503	86.594	118.944	224.585	—	253.876	4.866
		1000	11.087	22.350	96.913	178.994	229.357	489.315	—	419.488	8.334
4	10	100	16.768	17.611	215.934	234.204	187.441	235.970	59.010	30.493	0.878
	12	100	280.399	308.157	—	—	—	—	—	2787.256	35.755

Table 4.4: Computation times for strategies S1, . . . , S4 without and with neighborhood search on c/u instances

Opportunity of neighborhood search

Table 4.4 shows the computation times for strategies S1, S2, and S3 without and with neighborhood search, as well as the time spent on neighborhood search. As we can see, neighborhood search does not help much for the ranking and hybrid strategies. Indeed, all points computed from neighborhood search are quickly obtained by the ranking algorithm. However, neighborhood search provides a refined initial search region and therefore reduce the size of the ranking tree. This made it possible to solve instances of type $c/u/400/2/100$ using the ranking strategy S1. For the branch and bound strategy, the use of neighborhood search is clearly crucial.

Besides we observe from the results that our implementation of neighborhood search is very time-efficient.

Approximation in the ranking strategy

We present some results for approximation versions of the ranking strategy S1. We consider both versions ("A" and "B") described in Section 4.3.2 and 3 tolerance levels: $\varepsilon = 1\%$, $\varepsilon = 5\%$, and $\varepsilon = 10\%$.

Table 4.5 presents computation times for the exact version and the approximation versions. We guessed in Section 4.3.2 that version B could be more time-efficient than version A. As we can see, this is only the case on certain instances for $\varepsilon = 10\%$. For smaller tolerances, version B maintains a set of points N which is nearly as large as in the exact version and thus the procedure `updateSearchRegion` takes much more time than in version A.

Table 4.6 gives the output sizes. As expected, version A produces smaller output than version B, up to 74 times smaller than the nondominated set in the case $\varepsilon = 10\%$.

It turns out that for most tested bi-objective instances, the set of all extreme nondomi-

						$\varepsilon = 1\%$		$\varepsilon = 5\%$		$\varepsilon = 10\%$	
t	r	p	n	K	Exact	A	B	A	B	A	B
c	u	2	200	1000	10.432	0.719	0.674	0.634	0.646	0.647	0.663
			400	1000	56.463	8.663	12.811	8.832	8.606	8.747	8.626
		3	15	100	0.575	0.304	0.379	0.178	0.122	0.088	0.092
				1000	0.883	0.337	0.470	0.231	0.281	0.293	0.082
			20	100	5.759	1.475	2.786	0.344	0.351	0.198	0.106
				1000	11.087	1.578	4.152	0.449	1.136	1.047	0.206
		4	10	100	16.768	14.069	14.951	5.365	4.597	2.441	1.167
			12	100	280.399	214.333	370.275	27.178	37.111	5.224	5.342
	c	2	10	100	41.018	30.899	23.837	0.550	0.000	0.003	0.000
			12	100	155.810	—	—	0.002	0.000	0.000	0.000

Table 4.5: Comparison of computation times (s) for exact and approximate versions of the ranking strategy S1

nated points computed at the beginning of Algorithm 4.1 is nearly sufficient to ε-dominate all feasible points.

Finally, Table 4.7 provides two indicators for the output Y_ε of the approximation version of S1. We computed the a posteriori covering tolerance ε_c, i.e. the minimal value such that any feasible point is ε_c-dominated by some point of Y_ε. We also computed a stability tolerance ε_s as a measure of distribution, i.e. the maximal value such that no two points of Y_ε ε_s-dominate each other. As we can see, version A achieves an ε_c which is very close to the a priori ε for three and four objective instances.

Conclusions

We proposed in this chapter a hybrid approach which solves MOCO problems and is not restricted to the bi-objective case. The method is based on a ranking strategy that integrates into a branching scheme. We presented an instantiation on the MOST problem using efficient implementations of procedures for ranking spanning trees and obtaining a good initial set of points through neighborhood search. We showed that the proposed approach can be adapted to generate an approximate nondominated set. According to presented computational experiments, there is no dominating exact strategy. The ranking strategy is the most time-efficient for solving instances where the nondominated points are not too far from being supported and where the part of the set of feasible points around the Pareto front is not too dense. It outperforms branch and bound approaches on such instance types. It requires however much memory to solve large instances. The hybrid strategy is the most efficient for instances where good choices on a few branching edges makes it possible to obtain far easier subinstances. It is also interesting for large instances due to its memory save compared to the ranking only strategy. The branch and bound strategy

						$\varepsilon = 1\%$		$\varepsilon = 5\%$		$\varepsilon = 10\%$	
t	r	p	n	K	Exact	A	B	A	B	A	B
c	u	2	200	1000	12568	483	487	459	620	459	620
			400	1000	24476	1020	1020	1019	1379	1019	1379
		3	15	100	2844	1339	2822	261	1747	107	932
				1000	3555	1393	3489	265	2033	107	1168
			20	100	11195	3464	10579	472	3497	205	1277
				1000	15009	3528	13516	465	4540	204	1801
		4	10	100	3047	1939	3047	420	2709	159	2172
			12	100	17195	8838	17171	1172	9864	444	4140
	c	2	10	100	143	39	143	15	55	15	15
			12	100	246	—	—	19	23	18	18

Table 4.6: Number of points obtained for exact and approximate versions of the ranking strategy S1

					$\varepsilon = 1\%$				$\varepsilon = 5\%$				$\varepsilon = 10\%$			
					A		B		A		B		A		B	
t	r	p	n	K	ε_c	ε_s	ε_c	ε_s	ε_c	ε_s	ε_c	ε_s	ε_c	ε_s	ε_c	ε_s
c	u	2	200	1000	0.991	0.059	0.975	0.051	1.561	0.059	1.119	0.046	1.561	0.059	1.119	0.046
			400	1000	0.894	0.031	0.894	0.029	0.955	0.031	0.769	0.022	0.955	0.031	0.769	0.022
		3	15	100	1.000	0.252	0.635	0.133	4.986	0.554	4.078	0.136	9.846	1.145	8.251	0.144
				1000	1.000	0.112	0.657	0.015	4.980	0.453	3.985	0.015	9.890	1.131	8.434	0.018
			20	100	1.000	0.157	0.837	0.096	4.999	0.426	4.474	0.099	9.970	0.586	8.747	0.106
				1000	1.000	0.072	0.855	0.011	4.991	0.256	4.364	0.011	9.938	0.659	8.845	0.019
		4	10	100	0.999	0.520	0.070	0.199	4.993	1.181	2.780	0.199	9.981	1.423	6.198	0.199
			12	100	1.000	0.269	0.393	0.146	5.000	0.764	3.971	0.148	9.973	0.965	8.048	0.157
	c	2	10	100	0.963	0.430	0.000	0.151	3.886	0.400	2.736	0.153	4.626	0.400	4.070	0.371
			12	100	—	—	—	—	3.575	0.267	3.412	0.169	7.641	0.267	6.337	0.267

Table 4.7: Covering and stability tolerances (%) measured on the output sets of the approximate versions of the ranking strategy S1

remains powerful for hard instances containing nondominated points that are far away from the convex hull of feasible points and for which it is not sufficient to branch on a few edges. The implementation of neighborhood search we proposed is very efficient in practice. The two approximation versions we proposed run significantly faster than the exact version according to our experiments. The experiments also showed that the two versions do not dominate each other. However, with similar computation times, version A produces significantly smaller approximations.

There are several directions for future work. First it would be interesting to make the hybrid strategy more efficient on a wide range of instances. To this end, less time should be spent on each search node. The time spent on a search node is related to the number of ranking directions that are exploited and to the definition of the stopping criterion. It would be required in particular to avoid the use of ranking directions that yield redundant solutions. The stopping criterion should trigger a branching as soon as it is clear that the only use of ranking is too expensive. The literature on metaheuristics and genetic algorithms could provide some insight into this issue, since the tradeoff between the quality of a solution pool and the exploration cost is a concern (see e.g. Wagner et al., 2011). Future directions also include the instantiation of the hybrid approach on other MOCO problems, especially those for which an efficient ranking algorithm is available.

Conclusions

Summary

We considered in this thesis optimization with multiple objective functions, a subject lying at the intersection of classical operations research and multiple criteria decision making. We presented exact and approximate solution concepts as well as some solving approaches for MOCO problems. The latter revealed the encountered solving approaches can be classified in two categories, according to whether they primarily operate in the decision space or in the objective space. We also surveyed several aspects related to a particular MOCO problem, namely the multi-objective Minimum Spanning Tree problem.

We were concerned with the concept of *search region* in MOO. The subject was not much treated since the search region is easily represented in the bi-objective case. In the multi-objective case, the literature on exact solutions to MOO problems rarely addresses the problem of efficiently and explicitly representing the search region. The only proposal for an explicit representation of the search region, not based on solving integer programs, was found in Przybylski et al. (2010b). We proposed some enhancements of their algorithm as well as a new approach which avoids the generation of redundant elements. We studied complexity aspects related to the explicit representation of the search region and to its generation using the two approaches. We also compared the approaches on the basis of computational experiments. They showed that the two approaches behave similarly up to 5 objective functions and that the new approach outperforms the older above.

Besides, starting from the classification of MOCO solving methods we presented, we proposed a hybrid strategy which combines a ranking-based exploration of the objective space with an implicit enumeration scheme. The motivation behind this was to take advantage of the flexible choices that can be made in the decision space while also benefiting from a directed search in the objective space enabled by a ranking algorithm together with the weighted sum technique. The method was implemented and tested on several instances of the multi-objective Minimum Spanning Tree problem. The experimental results first show that instances with more than two objectives can be practically solved. The results did not reveal a dominating strategy on all instance types but provided some insight into the behavior of each strategy. We also showed that the proposed approaches can be adapted to quickly generate an approximate nondominated set with a priori quality guarantee.

Prospects

Among the perspectives we already drawn, we would like to highlight a few work directions which sound promising to us.

Exploitation of the neighborhood of search zones The properties we derived on local upper bounds addressed their relation to feasible points which contribute to define them. From this relation, we can define a neighborhood between local upper bounds, e.g. two local upper bounds being neighbors if they share the same defining points on one component. This neighborhood could be exploited in an enumeration MOCO approach. In particular, in the case where for each new point we know a search zone which contains it, we could exploit the neighborhood relation to possibly avoid the scan of all local upper bounds.

Parallel search strategies The exploration of distinct search zones as well as the exploration of distinct search nodes can be parallelized. It raises however the issue of concurrent updates of the search region.

Instantiation on other MOCO problems Other MOCO problems are known to have practically efficient ranking algorithms, namely Shortest Path and Knapsack as well as Assignment. Moreover, algorithms not limited to the bi-objective case have been developed for each of them. It would be interesting to compare the efficiency of a hybrid approach to state-of-the-art algorithms.

References

Aboudi, R. and Jørnsten, K. (1990). "Resource constrained assignment problems". In: *Discrete Applied Mathematics* 26.2–3, pp. 175–191. DOI: 10.1016/0166-218X(90)90099-X.

Aggarwal, V., Aneja, Y. P., and Nair, K. P. K. (1982). "Minimal spanning tree subject to a side constraint". In: *Computers & Operations Research* 9.4, pp. 287–296. DOI: 10.1016/0305-0548(82)90026-0.

Aggarwal, V. (1985). "A Lagrangean-relaxation method for the constrained problem". In: *Computers & Operations Research* 12.1, pp. 97–106. DOI: DOI:10.1016/0305-0548(85)90011-5.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. 1st. Prentice Hall.

Alonso, S., Domínguez-Ríos, M. Á., Colebrook, M., and Sedeño-Noda, A. (2009). "Optimality conditions in preference-based spanning tree problems". In: *European Journal of Operational Research* 198.1, pp. 232–240. DOI: DOI:10.1016/j.ejor.2008.07.042.

Andersen, K. A., Jörnsten, K., and Lind, M. (1996). "On bicriterion minimal spanning trees: An approximation". In: *Computers & Operations Research* 23.12, pp. 1171–1182. DOI: DOI:10.1016/S0305-0548(96)00026-3.

Aneja, Y. P. and Nair, K. P. K. (1979). "Bicriteria Transportation Problem". In: *Management Science* 25.1, pp. 73–78. DOI: 10.1287/mnsc.25.1.73.

Bazgan, C., Hugot, H., and Vanderpooten, D. (2009a). "Implementing an efficient fptas for the 0-1 multi-objective knapsack problem". In: *European Journal of Operational Research* 198.1, pp. 47–56. DOI: 10.1016/j.ejor.2008.07.047.

Bazgan, C., Hugot, H., and Vanderpooten, D. (2009b). "Solving efficiently the 0–1 multi-objective knapsack problem". In: *Computers & Operations Research* 36.1, pp. 260–279. DOI: 10.1016/j.cor.2007.09.009.

Belhoul, L., Galand, L., and Vanderpooten, D. (2014). "An efficient procedure for finding best compromise solutions to the multi-objective assignment problem ". In: *Computers & Operations Research* 0. DOI: 10.1016/j.cor.2014.03.016.

Bentley, J., Clarkson, K., and Levine, D. (1993). "Fast linear expected-time algorithms for computing maxima and convex hulls". In: *Algorithmica* 9.2, pp. 168–183. DOI: 10.1007/BF01188711.

Berg, M. de, Cheong, O., Kreveld, M. van, and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. 3rd. Santa Clara, CA, USA: Springer.

Boissonnat, J. D., Sharir, M., Tagansky, B., and Yvinec, M. (1998). "Voronoi Diagrams in Higher Dimensions under Certain Polyhedral Distance Functions". In: *Discrete & Computational Geometry* 19.4, pp. 485–519. DOI: 10.1007/PL00009366.

Borges, P. and Hansen, M. (2002). "A Study of Global Convexity for a Multiple Objective Travelling Salesman Problem". In: *Essays and Surveys in Metaheuristics*. Vol. 15. Springer US. Chap. Operations Research/Computer Science Interfaces Series, pp. 129–150. DOI: 10.1007/978-1-4615-1507-4_6.

Camerini, P. M., Galbiati, G., and Maffioli, F. (1983). "On the complexity of finding multi-constrained spanning trees ". In: *Discrete Applied Mathematics* 5.1, pp. 39–50. DOI: 10.1016/0166-218X(83)90014-8.

Clímaco, J. o. C. N. and Pascoal, M. M. B. (2012). "Multicriteria path and tree problems: discussion on exact algorithms and applications". In: *International Transactions in Operational Research* 19.1-2, pp. 63–98. DOI: 10.1111/j.1475-3995.2011.00815.x.

Corley, H. W. (1985). "Efficient spanning trees". In: *Journal of Optimization Theory and Applications* 45.3, pp. 481–485. DOI: 10.1007/BF00938448.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press.

Dächert, K. and Klamroth, K. (2014). "A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems". In: *Journal of Global Optimization*, pp. 1–34. DOI: 10.1007/s10898-014-0205-z.

Delort, C. and Spanjaard, O. (2010). "Using Bound Sets in Multiobjective Optimization: Application to the Biobjective Binary Knapsack Problem". In: *9th International Symposium on Experimental Algorithms (SEA 2010)*. Ed. by P. Festa. Vol. 6049. Lecture Notes in Computer Science. Springer Berlin Heidelberg. Chap. Lecture Notes in Computer Science, pp. 253–265. DOI: 10.1007/978-3-642-13193-6_22.

Delort, C. and Spanjaard, O. (2011). "Yet another two-phase method for the biobjective assignment problem". In: *21st International Conference on Multiple Criteria Decision Making (MCDM 2011)*. Jyvaskyla, Finland.

Deo, N. (1974). *Graph theory with applications to engineering and computer science*. Prentice-Hall.

Dhaenens, C., Lemesre, J., and Talbi, E. G. (2010). "K-PPM: A new exact method to solve multi-objective combinatorial optimization problems". In: *European Journal of Operational Research* 200.1, pp. 45–53. DOI: 10.1016/j.ejor.2008.12.034.

Diakonikolas, I. and Yannakakis, M. (2007). "Small Approximate Pareto Sets for Bi-objective Shortest Paths and Other Problems". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by M. Charikar, K. Jansen, O. Reingold, and J. P. Rolim. Vol. 4627. Springer Berlin Heidelberg. Chap. Lecture Notes in Computer Science, pp. 74–88. DOI: 10.1007/978-3-540-74208-1_6.

Ehrgott, M. (2005). *Multicriteria optimization*. Springer Verlag.

Ehrgott, M. (2008). "Multiobjective (Combinatorial) Optimisation - Some Thoughts on Applications". Anglais. In: *Multiple Objective Programming and Goal Programming: Theoretical Results and Practical Applications*. Ed. by V. Barichard, M. Ehrgott, X. Gandibleux, and V. T'Kindt. Vol. 618. Lecture Notes in Economics and Mathematical Systems. Tours, France: Springer, pp. 267–282. DOI: 10.1007/978-3-540-85646-7_25.

Ehrgott, M. and Gandibleux, X. (2007). "Bound sets for biobjective combinatorial optimization problems". In: *Computers and Operations Research* 34.9, pp. 2674–2694. DOI: 10.1016/j.cor.2005.10.003.

Ehrgott, M. and Klamroth, K. (1997). "Connectedness of efficient solutions in multiple criteria combinatorial optimization". In: *European Journal of Operational Research* 97.1, pp. 159–166. DOI: DOI:10.1016/S0377-2217(96)00116-6.

Ehrgott, M. and Wiecek, M. (2005). "Mutiobjective Programming". In: *Multiple Criteria Decision Analysis: State of the Art Surveys*. Ed. by J. Figueira, S. Greco, and M. Ehrgott. Vol. 78. Boston, Dordrecht, London: Springer Verlag. Chap. International Series in Operations Research & Management Science, pp. 667–708. DOI: 10.1007/0-387-23081-5_17.

Eppstein, D. (1998). "Finding the k Shortest Paths". In: *SIAM Journal on Computing* 28.2, pp. 652–673. DOI: 10.1137/S0097539795290477.

Eppstein, D. (1992). "Finding the k smallest spanning trees". In: *BIT Numerical Mathematics* 32.2, pp. 237–248. DOI: 10.1007/BF01994879.

Gabow, H. (1977). "Two Algorithms for Generating Weighted Spanning Trees in Order". In: *SIAM Journal on Computing* 6.1, pp. 139–150. DOI: 10.1137/0206011.

Gabow, H. and Myers, E. (1978). "Finding All Spanning Trees of Directed and Undirected Graphs". In: *SIAM Journal on Computing* 7.3, pp. 280–287. DOI: 10.1137/0207024.

Gabow, H. N. and Tarjan, R. E. (1985). "A linear-time algorithm for a special case of disjoint set union ". In: *Journal of Computer and System Sciences* 30.2, pp. 209–221. DOI: 10.1016/0022-0000(85)90014-5.

Gabow, H. N., Bentley, J. L., and Tarjan, R. E. (1984). "Scaling and Related Techniques for Geometry Problems". In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. STOC '84. New York, NY, USA: ACM, pp. 135–143. DOI: 10.1145/800057.808675.

Galand, L. (2008). "Méthodes exactes pour l'optimisation multicritère dans les graphes : recherche de solutions de compromis". In:

Galand, L. and Spanjaard, O. (2012). "Exact algorithms for OWA-optimization in multiobjective spanning tree problems ". In: *Computers & Operations Research* 39.7, pp. 1540–1554. DOI: 10.1016/j.cor.2011.09.003.

Galand, L., Perny, P., and Spanjaard, O. (2010). "Choquet-based optimisation in multiobjective shortest path and spanning tree problems". In: *European Journal of Operational Research* 204.2, pp. 303–315. DOI: 10.1016/j.ejor.2009.10.015.

Gorski, J., Klamroth, K., and Ruzika, S. (2011). "Connectedness of Efficient Solutions in Multiple Objective Combinatorial Optimization". In: *Journal of Optimization Theory and Applications* 150.3, pp. 475–497. DOI: 10.1007/s10957-011-9849-8.

Graham, R. L. and Hell, P. (1985). "On the History of the Minimum Spanning Tree Problem". In: *IEEE Ann. Hist. Comput.* 7.1, pp. 43–57. DOI: 10.1109/MAHC.1985.10011.

Grandoni, F., Ravi, R., and Singh, M. (2009). "Iterative Rounding for Multi-Objective Optimization Problems". In: chap. Iterative Rounding for Multi-Objective Optimization Problems, pp. 95–106. DOI: {10.1007/978-3-642-04128-0_9}.

Hamacher, H. (1995). "A note on K best network flows". In: *Annals of Operations Research* 57.1, pp. 65–72. DOI: 10.1007/BF02099691.

Hamacher, H. and Ruhe, G. (1994). "On spanning tree problems with multiple objectives". In: *Annals of Operations Research* 52.4, pp. 209–230. DOI: 10.1007/BF02032304.

Hansen, P. (1980). "Bicriterion Path Problems". In: *Multiple Criteria Decision Making Theory and Application*. Ed. by G. Fandel and T. Gal. Vol. 177. Lecture Notes in Economics and Mathematical Systems. Berlin: Springer Verlag. Chap. Multiple Criteria Decision Making Theory and Application, pp. 109–127. DOI: 10.1007/978-3-642-48782-8_9.

Hassin, R. and Levin, A. (2004). "An Efficient Polynomial Time Approximation Scheme for the Constrained Minimum Spanning Tree Problem Using Matroid Intersection". In: *SIAM Journal on Computing* 33.2, pp. 261–268. DOI: 10.1137/S0097539703426775.

Hong, S. P., Chung, S. J., and Park, B. H. (2004). "A fully polynomial bicriteria approximation scheme for the constrained spanning tree problem". In: *Operations Research Letters* 32.3, pp. 233–239. DOI: 10.1016/j.orl.2003.06.003.

Jiménez, V. M. and Marzal, A. (2003). "A Lazy Version of Eppstein's K Shortest Paths Algorithm". In: *Experimental and Efficient Algorithms*. Ed. by K. Jansen, M. Margraf, M. Mastrolilli, and J. D. P. Rolim. Vol. 2647. Springer Berlin Heidelberg. Chap. Lecture Notes in Computer Science, pp. 179–191. DOI: 10.1007/3-540-44867-5_14.

Jorge, J. (2010). "Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires". Français. PhD thesis. Université de Nantes.

Jouglet, A. and Carlier, J. (2011). "Dominance rules in combinatorial optimization problems ". In: *European Journal of Operational Research* 212.3, pp. 433–444. DOI: 10.1016/j.ejor.2010.11.008.

Kaplan, H., Rubin, N., Sharir, M., and Verbin, E. (2008). "Efficient Colored Orthogonal Range Counting". In: *SIAM Journal on Computing* 38.3, pp. 982–1011. DOI: 10.1137/070684483.

Katoh, N., Ibaraki, T., and Mine, H. (1981). "An Algorithm for Finding K Minimum Spanning Trees". In: *SIAM Journal on Computing* 10.2, pp. 247–255. DOI: 10.1137/0210017.

Kirlik, G. and Sayın, S. (2014). "A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems ". In: *European Journal of Operational Research* 232.3, pp. 479–488. DOI: 10.1016/j.ejor.2013.08.001.

Klamroth, K., Lacour, R., and Vanderpooten, D. (2014). *On the representation of the search region in multi-objective optimization*. Tech. rep. Université Paris-Dauphine.

Knowles, J. D. and Corne, D. W. (2001). "Benchmark Problem Generators and Results for the Multiobjective Degree-Constrained Minimum Spanning Tree Problem". In: *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. Morgan Kaufmann Publishers, pp. 424–431.

Knowles, J. D. and Corne, D. W. (2002). "Enumeration of Pareto optimal multi-criteria spanning trees – a proof of the incorrectness of Zhou and Gen's proposed algorithm ". In: *European Journal of Operational Research* 143.3, pp. 543–547. DOI: 10.1016/S0377-2217(01)00346-0.

Kung, H. T., Luccio, F., and Preparata, F. P. (1975). "On Finding the Maxima of a Set of Vectors". In: *Journal of the ACM* 22.4, pp. 469–476. DOI: 10.1145/321906.321910.

Laumanns, M., Thiele, L., and Zitzler, E. (2006). "An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method ". In: *European Journal of Operational Research* 169.3, pp. 932–942. DOI: 10.1016/j.ejor.2004.08.029.

Lokman, B. and Köksalan, M. (2013). "Finding all nondominated points of multi-objective integer programs". In: *Journal of Global Optimization* 57.2, pp. 347–365. DOI: 10.1007/s10898-012-9955-7.

Mares, M. (2008). "The saga of minimum spanning trees". In: *Computer Science Review* 2.3, pp. 165–221. DOI: 10.1016/j.cosrev.2008.10.002.

Mehlhorn, K. and Näher, S. (1990). "Dynamic fractional cascading". In: *Algorithmica* 5.1-4, pp. 215–241. DOI: 10.1007/BF01840386.

Miettinen, K. and Mäkelä, M. M. (2002). "On scalarizing functions in multiobjective optimization". In: *OR Spectrum* 24.2. 10.1007/s00291-001-0092-9, pp. 193–213. DOI: 10.1007/s00291-001-0092-9.

Murty, K. G. (1968). "Letter to the Editor—An Algorithm for Ranking all the Assignments in Order of Increasing Cost". In: *Operations Research* 16.3, pp. 682–687. DOI: 10.1287/opre.16.3.682.

Okamoto, Y. and Uno, T. (2011). "A polynomial-time-delay and polynomial-space algorithm for enumeration problems in multi-criteria optimization". In: *European Journal of Operational Research* 210.1, pp. 48–56. DOI: 10.1016/j.ejor.2010.10.008.

Özlen, M. and Azizoglu, M. (2009). "Multi-objective integer programming: A general approach for generating all non-dominated solutions". In: *European Journal of Operational Research* 199.1, pp. 25–35. DOI: 10.1016/j.ejor.2008.10.023.

Özpeynirci, O. and Köksalan, M. (2010). "An Exact Algorithm for Finding Extreme Supported Nondominated Points of Multiobjective Mixed Integer Programs". In: *Manage. Sci.* 56.12, pp. 2302–2315. DOI: 10.1287/mnsc.1100.1248.

Papadimitriou, C. H. and Yannakakis, M. (2000). "On the approximability of trade-offs and optimal access of Web sources". In: *41st IEEE Symposium on Foundations of Computer Science*, pp. 86–92. DOI: 10.1109/SFCS.2000.892068.

Pedersen, C. R., Nielsen, L. R., and Andersen, K. A. (2008). "An algorithm for ranking assignments using reoptimization". In: *Computers & Operations Research* 35.11, pp. 3714–3726. DOI: 10.1016/j.cor.2007.04.008.

Perny, P. and Spanjaard, O. (2005). "A preference-based approach to spanning trees and shortest paths problems****". In: *European Journal of Operational Research* 162.3, pp. 584–601. DOI: 10.1016/j.ejor.2003.12.013.

Preparata, F. P. and Shamos, M. I. (1985). *Computational geometry: an introduction*. New York, NY, USA: Springer-Verlag New York, Inc.

Przybylski, A., Gandibleux, X., and Ehrgott, M. (2008). "Two phase algorithms for the bi-objective assignment problem ". In: *European Journal of Operational Research* 185.2, pp. 509–533. DOI: 10.1016/j.ejor.2006.12.054.

Przybylski, A., Gandibleux, X., and Ehrgott, M. (2009). "Computational Results for Four Exact Methods to Solve the Three-Objective Assignment Problem". In: *Multiobjective Programming and Goal Programming*. Ed. by S. B. Heidelberg. Vol. 618. Springer Berlin Heidelberg. Chap. Computational Results for Four Exact Methods to Solve the Three-Objective Assignment Problem, pp. 79–88. DOI: 10.1007/978-3-540-85646-7_8.

Przybylski, A., Gandibleux, X., and Ehrgott, M. (2010a). "A Recursive Algorithm for Finding All Nondominated Extreme Points in the Outcome Set of a Multiobjective Integer Programme". In: *INFORMS Journal on Computing* 22.3, pp. 371–386. DOI: 10.1287/ijoc.1090.0342.

Przybylski, A., Gandibleux, X., and Ehrgott, M. (2010b). "A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives". In: *Discrete Optimization* 7.3, pp. 149–165. DOI: 10.1016/j.disopt.2010.03.005.

Raith, A. and Ehrgott, M. (2009). "A two-phase algorithm for the biobjective integer minimum cost flow problem". In: *Comput. Oper. Res.* 36.6, pp. 1945–1954. DOI: 10.1016/j.cor.2008.06.008.

Ralphs, T., Saltzman, M., and Wiecek, M. (2006). "An improved algorithm for solving biobjective integer programs". In: *Annals of Operations Research* 147.1, pp. 43–70. DOI: 10.1007/s10479-006-0058-z.

Ramos, R. M., Alonso, S., Sicilia, J., and González, C. (1998). "The problem of the optimal biobjective spanning tree ". In: *European Journal of Operational Research* 111.3, pp. 617–628. DOI: 10.1016/S0377-2217(97)00391-3.

Ravi, R. and Goemans, M. X. (1996). "The constrained minimum spanning tree problem". In: *Algorithm theory: SWAT'96: 5th Scandinavian Workshop on Algorithm Theory, Reykjavík, Iceland, July 3-5, 1996: proceedings*. Ed. by R. Karlsson and A. Lingas. Vol. 1097. Springer Berlin Heidelberg. Chap. Lecture Notes in Computer Science, pp. 66–75. DOI: 10.1007/3-540-61422-2_121.

Roy, B. (1985). *Méthodologie multicritère d'aide à la décision*. Paris: Economica.

Ruzika, S. and Hamacher, H. (2009). "A Survey on Multiple Objective Minimum Spanning Tree Problems". In: *Algorithmics of Large and Complex Networks*. Ed. by J. Lerner, D.

Wagner, and K. Zweig. Vol. 5515. Springer Berlin Heidelberg. Chap. Lecture Notes in Computer Science, pp. 104–116. DOI: 10.1007/978-3-642-02094-0_6.

Serafini, P. (1987). "Some Considerations about Computational Complexity for Multi Objective Combinatorial Problems". In: *Recent advances and historical development of vector optimization*. Ed. by J. Jahn and W. Krabs. Vol. 294. Lecture Notes in Economics and Mathematical Systems. Berlin: Springer-Verlag. Chap. Lecture Notes in Economics and Mathematical Systems, pp. 222–232. DOI: 10.1007/978-3-642-46618-2_15.

Sourd, F. and Spanjaard, O. (2008). "A multi-objective branch-and-bound framework. Application to the bi-objective spanning tree problem". In: *INFORMS Journal on Computing* 20.3, pp. 472–484. DOI: 10.1287/ijoc.1070.0260.

Steiner, S. and Radzik, T. (2003). *Solving the biobjective minimum spanning tree problem using a k-best algorithm*. Tech. rep. TR-03-06, Department of Computer Science, King's College London.

Steiner, S. and Radzik, T. (2008). "Computing all efficient solutions of the biobjective minimum spanning tree problem". In: *Comput. Oper. Res.* 35.1, pp. 198–211. DOI: http://dx.doi.org/10.1016/j.cor.2006.02.023.

Sylva, J. and Crema, A. (2004). "A method for finding the set of non-dominated vectors for multiple objective integer linear programs ". In: *European Journal of Operational Research* 158.1, pp. 46–55. DOI: 10.1016/S0377-2217(03)00255-8.

Sylva, J. and Crema, A. (2007). "A method for finding well-dispersed subsets of non-dominated vectors for multiple objective mixed integer linear programs ". In: *European Journal of Operational Research* 180.3, pp. 1011–1027. DOI: 10.1016/j.ejor.2006.02.049.

Sylva, J. and Crema, A. (2008). "Enumerating the set of non-dominated vectors in multiple objective integer linear programming". In: *RAIRO-Operations Research* 42.3, pp. 371–387. DOI: 10.1051/ro:2008018.

Tarjan, R. E. (1983). *Data structures and network algorithms*. Society for Industrial and Applied Mathematics.

Tarjan, R. E. (1979). "Applications of Path Compression on Balanced Trees". In: *J. ACM* 26.4, pp. 690–715. DOI: 10.1145/322154.322161.

Tenfelde-Podehl, D. (2003). *A recursive algorithm for multiobjective combinatorial problems with*. Tech. rep. Karl-Franzens-Univ. Graz & Techn. Univ. Graz.

Ulungu, E. L. and Teghem, J. (1995). "The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems". In: *Foundations of Computing and Decision Sciences* 20.2, pp. 149–165.

Vassilvitskii, S. and Yannakakis, M. (2005). "Efficiently computing succinct trade-off curves". In: *Theor. Comput. Sci.* 348.2, pp. 334–356. DOI: 10.1016/j.tcs.2005.09.022.

Villarreal, B. and Karwan, M. H. (1981). "Multicriteria integer programming: A (hybrid) dynamic programming recursive approach". In: *Mathematical Programming* 21.1, pp. 204–223. DOI: 10.1007/BF01584241.

Vincent, T., Przybylski, A., and Gandibleux, X. (2011). "Two phase method for Biobjective Mixed 0-1 Linear Programming". In: *21st International Conference on Multiple Criteria Decision Making, Jyväskylä, Finland*.

Visée, M., Teghem, J., Pirlot, M., and Ulungu, E. L. (1998). "Two-phases Method and Branch and Bound Procedures to Solve the Bi–objective Knapsack Problem". In: *Journal of Global Optimization* 12.2, pp. 139–155. DOI: 10.1023/A:1008258310679.

Wagner, T., Trautmann, H., and Martí, L. (2011). "A Taxonomy of Online Stopping Criteria for Multi-Objective Evolutionary Algorithms". In: *Evolutionary Multi-Criterion Optimization*. Ed. by R. C. Takahashi, K. Deb, E. Wanner, and S. Greco. Vol. 6576. Springer Berlin Heidelberg. Chap. Lecture Notes in Computer Science, pp. 16–30. DOI: 10.1007/978-3-642-19893-9_2.

White, D. J. (1986). "Epsilon efficiency". In: *Journal of Optimization Theory and Applications* 49.2, pp. 319–337. DOI: 10.1007/BF00940762.

Wierzbicki, A. P. (1986). "On the completeness and constructiveness of parametric characterizations to vector optimization problems". In: *Operations-Research-Spektrum* 8.2, pp. 73–87. DOI: 10.1007/BF01719738.

Willard, D. E. and Lueker, G. S. (1985). "Adding Range Restriction Capability to Dynamic Data Structures". In: *Journal of the ACM* 32.3, pp. 597–617. DOI: 10.1145/3828.3839.

Zhou, G. and Gen, M. (1999). "Genetic algorithm approach on multi-criteria minimum spanning tree problem". In: *European Journal of Operational Research* 114.1, pp. 141–152. DOI: 10.1016/S0377-2217(98)00016-2.

List of Algorithms

1.1 Computation of the set Y_{extr} in the bi-objective case (Aneja and Nair (1979)) 24

2.1 Generalization of Prim's algorithm by Corley (1985) 40
2.2 A Kruskal-like multi-objective spanning tree algorithm by Serafini (1987) . . . 41

3.1 Generic method to generate all nondominated points of a MOCO problem based on the definition of search zones 58
3.2 Update procedure of an upper bound set based on redundancy elimination . 60
3.3 Update procedure of an upper bound set based on redundancy elimination with an enhanced filtering step 62
3.4 Update procedure of an upper bound set based on the avoidance of redundancies : *SA case* 69
3.5 Update procedure of an upper bound set based on the avoidance of redundancies : *general case* 70

4.1 Ranking procedure 82
4.2 Main hybrid procedure 88
4.3 Procedure `firstStage` 89
4.4 Procedure `exploreSearchNode` 89
4.5 Implementation of a neighborhood search procedure for the MOST problem . 95
4.6 Procedure `updateSearchRegion` 96

Nomenclature

Abbreviations

MCDM	multiple criteria decision making, page 7
MCDA	multiple criteria decision aid, page 7
MOO	multi-objective optimization, page 7
MOCO	multi-objective combinatorial optimization, page 7
MOBB	multi-objective branch and bound, page 13
MOST	multi-objective Minimum Spanning Tree problem, page 33

Convenient notations

(z_j, a_{-j})	$(a_1, \ldots, a_{j-1}, z_j, a_{j+1}, \ldots, a_p)$, page 52

General problem notations

$\mathcal{X}$, x	decision space, solution (possibly infeasible), page 14
$\mathcal{Z}$, z	objective space, point, page 14
m	dimension of the decision space, page 14
p	dimension of the objective space, page 14
c_j, $c = (c_1, \ldots, c_p)$	objective function, vector of objective functions, page 14
X	set of all feasible solutions, page 14
Y	set of all feasible points, page 14

Pareto dominance and remarkable sets and points

$= \leqq < \leq$	dominance relations defined on $\mathcal{Z}$, page 14
$\approx \precapprox \prec \precsim$	dominance relations defined on $\mathcal{X}$, page 14
$\leqq_\varepsilon$	ε-dominance relation in the objective space, page 15
$\precapprox_\varepsilon$	ε-dominance relation in the decision space, page 15
Y_{nd}	set of all nondominated points, page 16
Y_{wnd}	set of all weakly nondominated points, page 16
X_{eff}	set of all efficient solutions, page 16
X_{weff}	set of all weakly efficient solutions, page 16
N	stable subset of $\mathcal{Z}$ w.r.t. $\leq$, page 16
z^{I}	ideal point, page 16
z^{N}	nadir point, page 16

Convex weighted sum scalarization

λ	weight vector of $\mathbb{R}^p_{\geq}$, page 17
c_λ	weighted sum scalarized function $\sum_{j=1}^{p} \lambda_j \cdot c_j$ associated to weight vector λ, page 17
$\mathrm{conv}(Q)$	convex hull of Q, page 17
$\mathrm{conv}(Q)_{\leqq}$	$\mathrm{conv}(Q) + \mathbb{R}^p_{\leqq}$, page 18
Y_{supp}	set of all supported points, page 18
X_{supp}	set of all supported solutions, page 18
Y_{extr}	set of all extreme nondominated points, page 18
X_{extr}	set of all extreme efficient solutions, page 18

Around the search region

$Z = (m, M)^p$	search interval (contains all feasible points), page 52
$S(N)$	search region associated to a stable set N, page 53
$U(N)$, u	upper bound set associated to a stable set N, local upper bound, page 53
$C(u)$	search zone associated to a local upper bound u, i.e. $C(u) = \{z \in Z : z < u\}$, page 53

Implicit enumeration

s	search node, page 19
r	root node, page 19
$X(s)$	set of extensions of a search node s, page 19
$Y(s)$	image in the objective space of set of extensions of a search node s, page 19

Résumé

De nombreux problèmes décisionnels nécessitent la prise en compte d'objectifs multiples souvent conflictuels. L'ensemble des solutions efficaces d'un tel problème, c'est-à-dire qui ne peuvent être améliorées simultanément sur tous les objectifs, peut être de grande taille. Pourtant, il peut être intéressant de déterminer cet ensemble dans son intégralité, et plus particulièrement l'ensemble des points non dominés, c'est-à-dire son image dans l'espace des objectifs. On s'attache dans cette thèse à plusieurs aspects liés à la résolution de problèmes multi-objectifs, sans se limiter au cas biobjectif. Nous considérons la résolution exacte, dans le sens de la détermination de l'ensemble des points non dominés, ainsi que la résolution approchée dans laquelle on cherche une approximation de cet ensemble dont la qualité est garantie a priori.

Nous nous intéressons d'abord au problème de la détermination d'une représentation explicite de la région de recherche. La région de recherche, étant donné un ensemble de points réalisables connus, exclut la partie de l'espace des objectifs que dominent ces points et constitue donc la partie de l'espace des objectifs où les efforts futurs doivent être concentrés dans la perspective de déterminer tous les points non dominés.

Puis nous considérons le recours aux algorithmes de séparation et évaluation ainsi qu'aux algorithmes de ranking afin de proposer une nouvelle méthode hybride de détermination de l'ensemble des points non dominés. Nous montrons que celle-ci peut également servir à obtenir une approximation de l'ensemble des points non dominés. Cette méthode est implantée pour le problème de l'arbre couvrant de poids minimal. Les quelques propriétés de ce problème que nous passons en revue nous permettent de spécialiser certaines procédures et d'intégrer des prétraitements spécifiques. L'intérêt de cette approche est alors soutenu à l'aide de résultats expérimentaux.

Mots-clés optimisation multi-objectifs, représentation de la région de recherche, algorithmes de séparation et évaluation, algorithmes de ranking, problème de l'arbre couvrant de poids minimal

Abstract

Many decision problems require to consider several, often conflicting, objectives. For such a problem, the set of all efficient solutions, i.e. that cannot be improved simultaneously on all objectives, can be large. Yet it may be interesting to generate this set or rather its image in the objective space known as the set of all nondominated points. This thesis deals with several aspects related to solving multi-objective problems, without restriction to the bi-objective case. We consider exact solving, which generates the nondominated set, and approximate solving, which computes an approximation of the nondominated set with a priori guarantee on the quality.

We first consider the determination of an explicit representation of the search region. The search region, defined with respect to a set of known feasible points, excludes from the objective space the part which is dominated by these points. Future efforts to find all nondominated points should therefore be concentrated on the search region.

Then we review branch and bound and ranking algorithms and we propose a new hybrid approach for the determination of the nondominated set. We show how the proposed method can be adapted to generate an approximation of the nondominated set. This approach is instantiated on the minimum spanning tree problem. We review several properties of this problem which enable us to specialize some procedures of the proposed approach and integrate specific preprocessing rules. This approach is finally supported through experimental results.

Keywords multi-objective optimization, representation of the search region, branch and bound algorithms, ranking algorithms, minimum spanning tree problem