PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'Université Paris-Dauphine

Multiobjective Optimization for Complex Systems

Soutenue par

Antoine KERBÉRÉNÈS

Le 03/12/2021

École doctorale n°543

Ecole Doctorale SDOSE

Spécialité

Informatique

Composition du jury :

Laetitia JOURDAN Professeure, Université de Lille	*Présidente du jury*
Kathrin KLAMROTH Professeure, Universität Wuppertal	*Rapporteure*
Dominique QUADRI Professeure, Université Paris-Saclay	*Examinatrice*
Vincent T'KINDT Professeur, Université de Tours	*Rapporteur*
Daniel VANDERPOOTEN Professeur, Université Paris Dauphine	*Directeur de thèse*
Jean-Michel VANPEPERSTRAETE Ingénieur, NAVAL GROUP Research	*Co-encadrant*

Dauphine | PSL
UNIVERSITÉ PARIS

À René Kerbérénès,

Remerciements

Je remercie tout d'abord Daniel Vanderpooten d'avoir accepté d'encadrer mes travaux de thèse, et je le remercie pour ses contributions, ses suggestions, sa disponibilité, sa minutie et sa patience pendant ces quatre années.

Je remercie Jean-Michel Vanpeperstraete de m'avoir proposé un sujet de thèse aussi passionnant qu'ouvert, me laissant explorer une variété d'approches, sanctionnées par des critères de performance clairs. Je remercie également Jean-Michel pour ses nombreuses suggestions, pour nos discussions philosophiques, littéraires et cinématographiques, ainsi que pour l'honnêteté de ses conseils tant relativement à la conduite des travaux qu'à mon orientation dans un monde post-doctoral dans lequel j'ai bien peu de repères.

Je remercie Kathrin Klamroth et Vincent T'Kindt d'avoir accepté d'être rapporteurs à ma soutenance de thèse, et de se consacrer à nouveau à une lecture constructive de mon manuscrit. J'ai bénéficié lors de la présoutenance de leurs remarques bienveillantes et de conseils dont j'ai taché de tirer parti pour terminer ce document. Je remercie également Dominique Quadri et Laetitia Vermeulen-Jourdan de l'intérêt qu'elles ont témoigné à ce sujet de thèse en acceptant d'être membres du jury de soutenance.

Je remercie ensuite Hélène et Marc Kerbérénès, mes parents, pour leur soutien tout au long d'un parcours d'études long et aux revirements parfois difficiles à justifier. Je remercie Dana Mukanova d'être à mes côtés chaque jour, de m'avoir convaincu de venir étudier à Paris, de m'avoir soutenu quand j'ai connu des échecs. Je la remercie pour sa bienveillance, son intransigeance et son bon goût en toutes choses.

Je remercie Satya Tamby pour son aide et le témoignage de son expérience. Je remercie Thomas Pontoizeau pour les mêmes raisons ainsi que pour sa complicité tant intellectuelle qu'artistique, et j'ai hâte de le retrouver sur scène. Je remercie Alex Savatovsky pour la série de mardi après-midi que nous avons passés à discuter, débattre et définir des sujets de recherche parfois sérieux, nous aventurant dans des régions situées de part et d'autre de la limite de nos domaines de compétence respectifs.

Je tiens à dire ma grattitude et le respect que je porte aux instituteurs et professeurs qui m'ont encouragé et orienté tout au long de ma scolarité et de mes études : Laurent Schnebelen, M. Jourdain, M. Capitaine, Yves Rocher, Vincent Piquemal, Cédric Brun, Mickaël Cozic et Jérôme Lang.

Enfin, mes amitiés fraternelles et mon allégeance définitive vont à Arthur Brière, Elliott Lancaster-Kheireddine et Guillaume Sajus, braves parmi les braves et gardiens intangibles de mon sens des priorités.

Abbreviations

MO	Multiobjective Optimization
MOP	Multiobjective Optimization Problem
IMOP	Integer variable Multiobjecitve Optimization Problem
BIMOP	Binary variable (or $0-1$) MOP
MOCO	Multiobjective Combinatorial Optimization Problem
GCP	Generic Coupled Problem
GUCP	Generic Uncoupled Problem
MS	Minkowski (set) sum
NDMSP	Problem of computing the non-dominated subset of a MS
IF	Intermediary Dominance Filtering
SqIF	Sequential Pooling Algorithm with IF
UPool	Unidirectional Pooling Algorithm
(MO)-REF	(Multiobjective) REF problem, as defined in C.4.1.1
DP	Dynamic Programming
KP	Knapsack Problem
MKP	Multiknapsack Problem
MCKP	Multichoice Knapsack Problem
GAP	Generalized Assignment Problem
CT	Computing Time
TG	Time Gain, as defined in 2.2.1
RS(LB)	Restriction of a GCP obtained by fixing coupled variables of all $s \in S$ except $\sigma(k)$ for each $k \in K$ (and associated lower bound set) as defined in 3.2.2
CS(UB)	*Copy-split* relaxation of a GCP (and associated upper bound set) as defined in C.3.3.1
OC(UB)	Relaxation of coupled variables in local constraints of a GCP (and associated upper bound set) as defined in C.3.3.2

Notations

$\mathbb{R}, \mathbb{Z}, \mathbb{N}$	sets of real numbers, relative integers, and positive integers
$p \in \mathbb{N}$	the number or objectives, or of decision criteria
$\mathbb{R}^p, \mathbb{Z}^p, \mathbb{N}^p$	sets of p-dimensional real-valued and relative or positive integer-valued vectors
$I = \{1, ..., n\} \subseteq \mathbb{N}$	a set of integers
X	the feasible set of some optimization problem
Y	a set of points in the objective space
$x = (x_1, ..., x_n)$	an n-dimensional vector with component x_i, $i \in \{1, ..., n\}$.
$x = (x_i \mid i \in I)$	an n-dimensional vector presented as a family indexed by $I \subseteq N$.
$f(x) = (f_1(x), ..., f_p(x))$	a vector valued function
$f(X)$	The set of all images of elements of X by f, i.e. $f(X) = \{f(x) \in \mathbb{R} \mid x \in X\}$
$y \geqq y'$	if and only if $y_j \geqslant y'_j$ for any $j \in \{1, ..., p\}$ (y weakly dominates y')
$y \geq y'$	if and only if $y \geqq y'$ and $y \neq y'$ (y dominates y')
$y > y'$	if and only if $y > y'$ for any $j \in \{1, ..., p\}$ (y strongly dominates y')
$xR_f x' \ \forall R \in \{\geqq, \geq, >\}$	if and only if $f(x) R f(x')$.
$\mathcal{E}(X, f)$	the set of efficient solutions of X according to objective functions $f_1, ..., f_p$
$\tilde{\mathcal{E}}(X, f)$	a subset of $\mathcal{E}(X, f)$ which is isomorphic to $\mathcal{N}(f(X))$
$\mathcal{N}(Y)$	the set of non-dominated points in Y
$\mathbf{0}$	for some contextual $p \in \mathbb{N}$, a vector such that $\mathbf{0}_i = 0$ for all $i \in \{1, ..., p\}$
$t(x)[x_i \leftarrow a]$	a term t where x_i (either free or already valued) is assigned value a
x_{-i}	the subvector of x composed of all components except i, $x_{-i} = (x_1, ...x_{i-1}, x_{i+1}, ..., x_n)$
x_I	for $I \subseteq N$, the subvector of x composed of components with index in I.
$x_{\bar{I}}$	for $I \subseteq N$, the subvector of x composed of components with index in $N \backslash I$.
$x \sim (x_{-i}, x_i)$	x is equal up to some permutation to (x_{-i}, x_i)
$\mathbb{R}^p_{\geqq}, \mathbb{R}^p_{\geq}, \mathbb{R}^p_{>}$	resp. $\{y \in \mathbb{R}^p \mid y \geqq \mathbf{0}\}$, $\{y \in \mathbb{R}^p \mid y \geq \mathbf{0}\}$, $\{y \in \mathbb{R}^p \mid y > \mathbf{0}\}$
$\langle \lambda, y \rangle$	Inner product of $\lambda \in \mathbb{R}^p$ and $y \in \mathbb{R}^p$, $\langle \lambda, y \rangle = \sum_{j=1}^p \lambda_j y_j$

In *GUCP* and *GCP*:

N	a set of variable indices
$S \subseteq \mathcal{P}(N)$	a set of subsystems
$K \subseteq \mathcal{P}(N)$	a set of couplings
$\gamma(k) \subset S$	the set of subsystems coupled by coupling $k \in K$
X^k for $k \in K$	the feasible set defined by coupling constraints alone
X^s for $s \in S$	the feasible set defined by local constraints of system $s \in S$ alone

In *REF*:

T	a set of tasks
$T_C \subseteq T$	the subset of complex tasks
$T_L \subseteq T$	the subset of local tasks
M	a set of machines
S	a set of sites
$S(t)$	the subset of sites which can perform task $t \in T$
$T(s)$	the subset of tasks which can be performed on site $s \in S$
$M(s)$	the subset of machines available on site $s \in S$
$M(S(t))$	the subset of machines available to perform task $t \in T$ across all sites
$T_L(s) = T_L \cap T(s)$	the subset of tasks local to site $s \in S$
S^{T_C}	the set of all *decouplings* i.e. restrictions of each complex task $t \in T_C$ to some $s \in S$
$\sigma \in S^{T_C}$	a *decoupling*, i.e. a restriction of each complex task to some site $s \in S$.

Contents

Introduction **9**

1 Fundamental notions **13**

1.1 Notions of Multiobjective optimization 14

1.2 Coupled problems and decomposition 22

1.3 The Generic Uncoupled Problem and the efficiency of decomposition 28

1.4 Conclusions 31

2 Computing efficiently the non-dominated subset of the Minkowski set sum **33**

2.1 The non-dominated subset of the Minkowski sum problem (NDMSP) 34

2.2 Intermediary filtering 35

2.3 A unidirectional method for pooling 40

2.4 Box-based methods 47

2.5 Conclusions and discussion 56

3 Decoupling a coupled problem to obtain bound sets **59**

3.1 Introduction 60

3.2 Decomposable restrictions and lower bound sets 63

3.3 Decomposable relaxations and upper bound sets 65

3.4 Experimental results 68

3.5 Conclusions and discussion 74

4 Application Problem **77**

4.1 Presentation of the REF problem 78

4.2 Upper and lower bound sets for REF 87

4.3 "Bottom-up" approaches based on Dynamic Programming 98

4.4 "Top-down" approaches for solving REF 105

4.5 Conclusions and perspectives 120

General conclusions and perspectives **121**

Appendices **127**

A The double bound as a solution concept in MOCO **129**

A.1 Definitions 129

A.2 Experimental Results 131

B Weak lower bounds obtained from correcting weak upper bound solutions **133**

B.1 Correction of solutions to the copy-split relaxation 133

B.2 Correction of solutions to the relaxation of coupled variables in subsystems knapsack constraints 134

B.3 Experimental results 135

C Résumé en Français **137**

C.1 Notions fondamentales 137

C.2 Calcul efficace du sous ensemble non-dominé d'une somme d'ensembles 140

C.3 Obtenir des ensembles bornant par décomposition 144

C.4 Problème d'application 149

Introduction

Complex systems and their optimization

A complex system is a collection of subsystems, which are independent enough to be identified, but still linked together in significant ways. A group of agents who share the ability to perform some tasks, or share access to some resources, may be thought of as a complex system. Other examples may be a firm producing on multiple sites with partially overlapping capabilities, or a manufactured product made of several interacting components. In all of these cases, multiple subsystems are to act or be acted upon, and this action has some consequences on the state of the other subsystems.

When attempting to operate or design complex systems, a trade-off thus appears naturally between, on the one hand, *bottom-up* reasoning - trying to infer the optimal state of the system from the optimal states of subsystems obtained independently, and on the other hand, *top-down* reasoning - assuming the position of a central planner. The first type of approach is usually computationally easier than the second type, since individual subsystems are smaller than the whole system. However, by not taking their interactions fully into account, one may propose either infeasible or suboptimal solutions.

The main challenge of complex systems optimization is, therefore, to take into account the interrelations between subsystems, while never considering the problem of the whole system at once. In bottom-up approaches, it can be done implicitly by adding, to the representation of the state of a subsystem, information that accounts for some scenario of interaction with the other systems. Top-down approaches need not reduce themselves to solving the original formulation of the problem using a generic method. If interactions between subsystems can be described as discrete, and are in finite number, a top-down approach may attempt to enumerate them, and explore them in an efficient way. Finally, some approaches may appear as hybrid between bottom-up and top-down. Examples of hybrid approaches include iterative methods to achieve coordination between the subsystems: the state of the interaction is fixed, subsystem problems are solved independently, and the state of the interaction is then adapted, so as to improve the results of the next subsystems optimization.

Related work and contribution

The present study focuses on a representation of complex as *coupled systems*, in the sense that the interaction between subsystems is modeled by *coupling constraints* in the mathematical representation of the optimization problem. Manipulation of these constraints and of the variables which appear in them are key to the resolution of coupled system optimization problems. The goal of these manipulations is to reduce, as much as possible, the resolution of the original, coupled problem, to the resolution of a sequence of decoupled problems that can be solved independently.

In the literature on the optimization of complex systems, this is called decomposition and coordination. In such approaches, the state of the interaction can be represented explicitly, by an upper level problem dedicated to making strategic decision coordinating subproblems, as in multilevel optimization (Colson et al. (2007), Migdalas et al. (2013), Liu et al. (2021)). It can also be represented more implicitly by the value of a penalty term in the objective function, measuring the violation of coupling constraints, as in Lagrangian relaxation (see e.g. Bertsekas et al. (1995)).

We consider multiobjective integer optimization problems (MOIP), which distinguishes it from most of the classical literature on complex system optimization. Recent work has used multiobjective optimization as a device for the optimization of complex systems, see, e.g., Yildiz et al. (2018), Dietz et al. (2020). In these approaches, each subsystem problem can be either a single, or a multiobjective problem, and the objective space of the global problem is the product of the objective spaces of the subproblems. The interaction between subsystems is represented in a higher dimensional objective space in which the objective spaces of the subproblems are embedded. The goal of such approaches is to find trade-offs between *competing* subsystem optimizations, which is not what we pursue here. Rather, in our case, subsystems and the global system share the same objective space, and the objective functions of the global systems are additively separable into the objective functions of the subsystems, so that subsystems are not competing.

There exists, on the one hand, literature on the application of classical decomposition and coordination methods such as Lagrangian relaxation to integer programming, e.g., Fisher (1981), Lemaréchal (2001) or Geoffrion (2010). On the other hand, the multiobjective optimization of complex systems is a well developed topic in the literature on metaheuristics, especially genetic algorithms applied to multidisciplinary design. However, our work is aimed at exact methods, and most existing work in exact methods using decomposition in the multiobjective case is dedicated to continuous optimization (e.g., Nakayama (1984, 1985), TenHuisen and Wiecek (1994)) and the characterization of the efficiency of single solutions. Thus, although work from these streams of research could be built upon for the computation of supported solution to the linear relaxation of a discrete multiobjective problem, and may be adapted to approach supported points, they cannot be applied to enumerate all efficient solutions of a discrete multiobjective problem.

Some topics of interest in this study are related to work by Gardenghi (2009) and Gardenghi et al. (2011) on the algebra of efficient sets, and pertain to the combination of efficient solutions to multiobjective subsystem problems, as well as the conditions under which these operations can yield efficient solutions to the global problem. We explore this topic in the particular case of

combinatorial problems, and we study the algorithmic aspects of these operations in addition to their algebraic aspects.

Layout of the thesis

In chapter C.1, we recall basic notions of multiobjective optimization, with a particular focus on dominance filtering algorithms. We then formally introduce coupled problems, and we define the key concept of *decomposition*, in the special case of uncoupled subproblems. Finally, we show that in the uncoupled case, a simple implementation of decomposition already yields considerable performance increase.

In chapter 2, we study the issue of combining solutions from subproblems of a decomposable MOCO problem. We propose several ways of increasing the performance of this operation, which is fundamental to all resolution approaches subsequently developed.

In chapter 3, we propose generic multiobjective upper and lower bounds (called *bound sets*) to the non-dominated set of a coupled problem, and we show that they can be obtained by solving uncoupled variants of that problem. We detail several of these variants and evaluate them experimentally.

In chapter 4, we present an application problem: a multiobjective '*multi-site* assignment problem denoted by *REF*. After having introduced the formal framework of Dynamic Programming, we show that REF admits such a resolution method, and we show how decomposition can be used to improve this initial resolution method, in a "bottom-up" approach. We also show how notions of bound sets introduced in chapter 3 adapt to the resolution of REF, using dynamic programming. We then turn to a "top-down" approach to solving REF and show that in some cases, all possible ways to decouple the problem can be enumerated and solved using decomposition. We speed this "top-down" approach by using dynamic programming to solve subproblems, by the precomputation of parts of the problem shared by all decouplings, and by applying branch & bound tricks, using an original *decoupling branching scheme*.

Finally, our conclusions recall the contributions made in each chapter of the thesis, as well as limitations of our results.

Acknowledgement

This work was partly funded by the French National Agency of Research and Technology (ANRT), CIFRE grant number 2017/1000, and supported by NAVAL GROUP RESEARCH.

Chapter 1

Fundamental notions

Chapter Abstract

In this chapter, we begin with recalling basic notions of multiobjective optimization, and an overview of dominance filtering algorithms. We then introduce the key concepts in this thesis: complex system optimization problem, uncoupled problems and decomposition. We define a *generic coupled problem* and a *generic uncoupled problem*, and illustrate the power of decomposition as a means to solve generic uncoupled problems.

Contents

1.1 Notions of Multiobjective optimization 14
1.1.1 Notations 14
1.1.2 The specific difficulties of $0-1$ integer MO 16
1.1.3 Dominance filtering algorithms 17
1.1.4 Choosing a dominance filtering algorithm 22
1.2 Coupled problems and decomposition 22
1.2.1 Decomposable formulations 23
1.2.2 Uncoupled optimization 25
1.3 The Generic Uncoupled Problem and the efficiency of decomposition . . . 28
1.4 Conclusions 31

1.1 Notions of Multiobjective optimization

A multiobjective optimization problem (MOP) is the problem of optimizing a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^p$, for $n, p \in \mathbb{N}$. We will call an element of the domain of f a *solution*, and its image by f a *point*. In order to optimize a vector-valued function, one must specify the order relation, the minima or maxima of which are to be computed. Usually, the component-wise order over points, called the *dominance* relation, is used. It is only a partial order, and the set of optimal elements of chains in this partial order defines the most common solution concept of multiobjective optimization: the *Pareto* set, or *efficient* set, and its image by f: the set of *non-dominated* points.

In applicative contexts, the optimization of a vector-valued function is meant to solve a multicriteria decision problem. Thus, to the p dimensions of co-domain of f correspond p objective functions which are usually given independently and explicitly. It will generally be assumed that no point is optimal for all p functions, but rather, that the objectives are somewhat conflicting and cannot be optimized simultaneously.

1.1.1 Notations

For $N = \{1, ..., n\}$ a set of variable indices, let $f : \mathbb{R}^n \to \mathbb{R}^p$ be a vector-valued function and $X \subseteq \mathbb{R}^n$ be the feasible set. Thus, any feasible solution $x = (x_1, ..., x_n) \in X$ is evaluated by $f(x) = (f_1(x), ..., f_p(x))$. Accordingly, the multiobjective optimization problem (MOP) associated with function f and feasible set X can be written as:

$$\begin{aligned} \max \quad & (f_1(x), ..., f_p(x)) \\ & x \in X \end{aligned}$$

Feasible solutions to a MOP can be considered from the perspective of the *decision space* $\mathbb{R}^n$, as elements in X, or from the perspective of the objective space $\mathbb{R}^p$, as feasible points, i.e. elements of $f(X) = \{f(x) \mid x \in X\}$. We will write $Y := f(X)$ only when objective functions and solution sets are fixed. For any $x \in \mathbb{R}^p$, we will write $y := (y_1, ..., y_p) = f(x)$ for short. In the MOP literature, comparisons in the decision and objective spaces are expressed in the following terminology (in accordance with our application cases, we use maximization as the default case for dominance):

Definition 1.1. *Given $y, y' \in \mathbb{R}^p$, y is said to* dominate *y', written $y \succeq y'$, if y is at least as good as y' according to all objectives, and is strictly better than y' according to at least one objective. Formally:*

$$y \succeq y' \Leftrightarrow \begin{cases} \forall j \in \{1, ..., p\}, & y_j \geqslant y'_j \\ \exists k \in \{1, ..., p\}, & y_k > y'_k \end{cases}$$

Variants of dominance in the objective space include the case where points can be equal,

$$y \geqq y' \Leftrightarrow \forall j \in \{1, ..., p\}, \quad y_j \geqslant y'_j$$

and the case where strict inequalities are required on all objectives

$$y \succ y' \Leftrightarrow \forall j \in \{1, ..., p\}, \quad y_j > y'_j$$

Definition 1.2. *A point $y \in Y$ is* non-dominated *if and only if there is no $y' \in Y$ such that $y' \succeq y$, and is* weakly non-dominated *if and only if there is no $y' \in Y$ such that $y' \succ y$.*

Definition 1.3. *A solution $x \in X$ is* efficient *with respect to f if and only if there is no $x' \in X$, x' such that $f(x') \succeq f(x)$, and is* weakly efficient *if and only if there is no $x' \in X$ such that $f(x') \succ f(x)$.*

Thus, the usual solution concept for multiobjective optimization is the enumeration of either the set of efficient solutions defined as

$$\mathcal{E}(X, f) := \{x \in X \mid x \text{ is efficient with respect to } f\}$$

or the set of non dominated points defined as

$$\mathcal{N}(Y) := \{y \in Y \mid y \text{ is non-dominated}\}$$

in the latter case, the decision maker resorting to this solution concept is assumed to be indifferent between solutions sharing the same value vector, while in the former case two solutions with the same value vector will be returned, and discrimination between those two, if needed, should be justified by other means. Finally, let us characterize a particular subset of efficient solutions.

Definition 1.4. *An efficient solution $x' \in \mathcal{E}(X, f)$ is supported if and only if there exists some $\lambda = (\lambda_j \mid j \in \{1, ..., p\}) \in \mathbb{R}^p_>$ such that x' is an optimal solution for*

$$\max_{x \in X} \sum_{j=1}^{p} \lambda_j f_j(x)$$

where $\mathbb{R}^p_> = \{a \in \mathbb{R}^p \mid a_j > 0, \forall j \in \{1, ..., p\}\}$. λ will be called a *weight vector*. A set of supported solutions can be found fairly easily by defining some policy for exploring the weight space. In the bi-objective case, a *dichotomic method* introduced by Aneja and Nair (1979) makes it possible to exhaustively and efficiently enumerate the set of supported solutions (see also Ulungu and Teghem (1995)). As is well known (see e.g. Ehrgott (2005)), all supported solutions are efficient, but not all efficient solutions are supported. It has furthermore been observed, e.g. by Visée et al. (1998) in the case of the bi-objective knapsack problem, that the number of unsupported solutions actually grows faster than the number of supported solutions as the instance size increases.

Finally, let us define the lexicographic order relation $\geq_{Lex}$, which is often used in the context

of multiobjective optimization, and will appear in several of the methods we develop or employ.

Definition 1.5. *For* $y = (y_1, y_2, ...y_p)$ *and* $y' = (y'_1, y'_2..., y'_p) \in \mathbb{R}^p$, $y \succeq_{Lex} y'$ *if and only if* $y_1 > y'_1$, *or* $y_1 = y'_1$ *and* $(y_2, ...y_p) \succeq_{Lex} (y'_2, ..., y'_p)$.

1.1.2 The specific difficulties of $0-1$ integer MO

We take special interest in problems with binary decision variables, i.e. problems which admit a formulation of the form:

$$\begin{aligned} \max \quad & (f_1(x), ..., f_p(x)) \\ & x \in X \\ & x_i \in \{0, 1\} \qquad\qquad \forall\, i \in \{1, ..., n\} \end{aligned}$$

Such problems may be called binary integer multiobjective problems (BIMOP). We focus on this type of problems because the application we study in chapter 4 is a generalized assignment problem, in which individual assignments are best described as binary variables.

Most problems studied in the BIMOP literature, which include multiobjective combinatorial optimization problems (MOCO), are NP-hard, even when their single-objective counterparts are of lower complexity (see, e.g., Serafini (1987)). Several aspects of this difficulty can be highlighted, which are of particular interest to our work.

In a multiobjective problem with sufficiently conflicting objectives, there are not one but many solutions to be outputted, according to the non-dominated set solution concept. It has been shown (see, e.g., Bazgan et al. (2013)) that the number of efficient solutions to an integer MOP (IMOP) grows exponentially with two factors. On the one hand, with the number of dimensions of the objective space, which impacts the filtering strength of the dominance relation: the higher the number of dimensions, the lower the likelihood that two random solutions will be comparable. Thus, algorithms requiring multiple dominance tests between feasible or partial solutions may be viable in the bi-objective case. But beyond, as dominance loses its ability to discriminate between solutions, one should be mindful of the size of sets one attemps to compute the efficient subset of. On the other hand in the case of IMOP, the discreteness of the decision space is reflected in the set of feasible points: the magnitude of the coefficients determines the number of possible vector values, and thus the sheer number of different points. This in turn affects the potential number of non-dominated points. When a MOP problem admits a family of instances for which the number of non-dominated points is exponential with respect to the number of variables, the problem is said to be *intractable* (Ehrgott (2005)). In this case, no algorithm can enumerate the non-dominated set in polynomial time with respect to the size of the instance. In particular, the multiobjective versions of combinatorial optimization problems like *shortest path, minimum spanning tree, assignment, knapsack, traveling salesperson* are intractable.

The current literature in IMOP algorithmics may roughly be divided between decision space

methods and objective space methods. The former focus on the constructive search of solutions, trying to shorten this search by eliminating as many as possible of the candidate partial solutions being constructed using bounding reasoning, comparisons between partial solutions, and between partial solutions and already complete and feasible solutions. Branch & Bound (see the survey by Przybylski and Gandibleux (2017) and Sourd and Spanjaard (2008)) and Dynamic Programing (see e.g. Klamroth and Wiecek (2000), Bazgan et al. (2009), Delort and Spanjaard (2013)) are examples of such decision space methods, and they tend to run into the first previously described difficulty: comparisons in higher dimension spaces are rarely possible.

Objective space methods, rather than constructing and comparing partial solutions, attempt to modify the original problem so that each non-dominated point can be obtained with as few calls to a single objective solver as possible. To this end, two main tools are used: scalarizations, and the addition of constraints. Whereas weighted sums can only yield supported points, more complex scalarizations, such as Tchebychev distance to a reference point, can yield any non-dominated point. This was the basis for some early generic integer MOP methods, such as Eswaran et al. (1989) and Sayın and Kouvelis (2005). To avoid the optimization of a nonlinear function such as Tchebychev distance and still deliver non-supported solutions, one can restrict the scalarized problem to subregions of the criterion space so that the desired unsupported solution becomes locally supported, or a solution to a local lexicographic optimization, or even the local optimization of one objective only. In the bi-objective case, this gives rise to the most competitive generic integer MOP methods: the e-constraint method, first introduced by Haimes et al. (1971). For more than two objectives, recent literature is seeing the rise of generic algorithms, e.g. generalizations of the ε-constraint method by Laumanns et al. (2006), Kirlik and Sayın (2014) and improvements on the constrained exploration of the objective space by Lokman and Köksalan (2013), Boland et al. (2017) as well as Tamby and Vanderpooten (2021), based on the notion of *search region* introduced by Klamroth et al. (2015).

1.1.3 Dominance filtering algorithms

1.1.3.1 Algorithms over linear list solution archives

The most basic of dominance filtering algorithms is one which iterates through a list of *input* points Y, comparing each $y \in Y$ to every point in $\widetilde{Y}$, a set of *candidate* points, i.e. points which have not yet been found to be dominated by any point in Y. For each $y' \in \widetilde{Y}$ such that $y \succeq y'$, y' is removed from $\widetilde{Y}$. If there is no $y' \in \widetilde{Y}$ such that $y' \succeq y$, then y is added to $\widetilde{Y}$. In the worst case, i.e. if Y is an independent set with respect to $\preceq$, this algorithm requires $|Y|^2$ dominance tests. In practice, *move to front*-type heuristics as introduced by Bentley et al. (1993) can be used to bring about conclusive dominace tests sooner in the loop over $\widetilde{Y}$.

The main heuristic described by Bentley et al. (1993) consists in moving the candidate point y' found to dominate some input y to the front of $\widetilde{Y}$. We denote it as *MCtF* for *Move Candidate To Front*. This is based on the intuition that the next input point should be compared first to candidates

found to dominate previous inputs. Points that are often found to dominate other points have good chances to be non-dominated, thus, comparing them early to each input should be profitable. Cornu (2017) interpreted this heuristic as also prescribing an insertion of a non-dominated input at the front of $\widetilde{Y}$, which we also consider, and denote by *MItF*, for *Move Incumbent to Front*.

The basic algorithm can also be improved upon very significantly by taking advantage of a prior ordering of the input set, as has been described by Kung et al. (1975). Indeed, if Y is sorted according to $\succeq_{Lex}$, a point cannot dominate any of its predecessors. Thus, half of the dominace tests required for filtering can be saved, as it is only necessary to check whether an input is dominated by no candidate. In the biobjective case, a crucial observation is that, if Y is sorted lexicographically, then input solution y is dominated by an element in $\widetilde{Y}$ if and only if it is dominated by the last element of $\widetilde{Y}$. We call a dominance filtering algorithm based on this principle a *unidirectional dominance filtering* algorithm (UF). Using the unidirectional algorithm described as Algorithm 7, *MCtF* can still be used if $p \geqslant 3$. However *MItF* becomes irrelevant, since an input point cannot dominate a candidate point.

Algorithm 1: Unidirectional DominanceFilter

input : $Y \subseteq \mathbb{R}^p$
output: $\mathcal{N}(Y)$

$Sort(Y, \succeq_{Lex})$
$\widetilde{Y} \leftarrow \{y^1\}$
/*The for loop iterates according to $\succeq_{Lex}$ order */
for $y \in Y$ **do**
  $Dominated \leftarrow False$
  if $p = 2$ **then**
    /*Assume $\widetilde{Y} = \{y^1, ..., y^m\}$ */
    if $y^m \succeq y$ **then**
      $Dominated \leftarrow True$
  else
    /*$p \geqslant 3$ */
    for $y' \in \widetilde{Y}$ **do**
      if $y' \succeq y$ **then**
        $Y \leftarrow moveToFront(Y, y')$
        $Dominated \leftarrow True$
  if $Dominated = False$ **then**
    $\widetilde{Y} \leftarrow Y \cup \{y\}$
    /*If $p = 2$, this means an insertion at the back. If $p \geqslant 2$, depending on the heuristic used, this could mean an insertion at the front, or at the back of $\widetilde{Y}$. */
return $\widetilde{Y}$

1.1.3.2 Algorithms over binary trees with p-dimensional intermediary node data (KDTrees)

The first type of alternative to list-based dominance filtering algorithms is filtering by insertion into a *KDTree* in the sense of Chen et al. (2012), which these authors incorporate into a *two phase* method of dominance filtering, which is described as Algorithm 2. We present this algorithm in details, because we will later adapt it to dominance filtering for sets of more complex objects than mere points.

Assume the points set is structured into a list, and initialize a second list of candidate points. Considering points in order, check whether each incoming point is dominated by one of the previously considered candidates. If it is not, add it to the list of candidates. It is now guaranteed that no point is dominated by its predecessors. Reverse the list. Now, by repeating the same operation, we will make sure that no remaining point is dominated by any of its former successors, and thus that no remaining point is dominated at all.

Algorithm 2: Two-phase dominance filtering algorithm using KD-Trees

input : $Y \subseteq \mathbb{R}^p$
output: $\mathcal{N}(Y)$
/*Assume that $Y = \{y^1, ...y^n\}$. */
Initialize $Tree^1$
$s \leftarrow y^1$
$k = 0$
for $y \in Y \backslash \{y^1\}$ **do**
  if $s \not\succeq y$ **then**
    if $||s||_2 < ||y||_2$ **then**
      $s \leftarrow y$
    if $Dominated(Tree^1, y) = False$ **then**
      $Insert(Tree^1, y)$
  if $k = \left\lfloor |Y|^\delta \right\rfloor$ **then**
    /*when $\left\lfloor |Y|^\delta \right\rfloor$ input points have been considered */
    $Pruning(Tree^1)$
  $k \leftarrow k + 1$
Initialize $Tree^2$
Assume $Content(Tree^1) = \{y^1, ..., y^m\}$
for $y \in \{y^m, ..., y^1\}$ **do**
  if $Dominated(Tree^2, y) = False$ **then**
    $Insert(Tree^2, y)$
return $Content(Tree^2)$

Chen et al. (2012) improve this method in three main ways. First, by embedding the lists built in each of the two phases into *KDTrees*. A KDTree is a binary tree, in which each node has therefore two children, denoted *LChild* and *RChild*, and is associated with a point $y \in \mathbb{R}^p$, and two bounds. u^y, l^y denote respectively the multiobjective upperbound and lower bound to the subtree

rooted in y, meaning that for all point y' in the subtree, $u^y \succeq y' \succeq l^y$. As is well known, binary trees allow one to speed up the verification that a point is non-dominated, by avoiding tests using bound reasoning, and Chen et al. (2012) use them in a classic fashion, described by Algorithm 3. If a point is found to be non-dominated, it is inserted in the KDTree using Algorithm 4.

Second, Chen et al. (2012) use a *sieve* point to try and perform first dominance tests which are likely to succeed. This is essentially the same trick as in *move-to-front* heurisitc. In this case the dominance power of a point is heuristically evaluated by its l^2 norm. For each input point y considered in the first phase, first check whether it is dominated by the *sieve* point, and if it has greater l^2 norm than the current sieve point, make y the new sieve point. Finally, Chen et al. (2012) use *pruning*, a common concept in search algorithms over tree-like structure: within the first phase of their algorithm, they sometimes check whether some nodes of the tree, seen as points, dominate previously inserted nodes, thus performing in advance some operation belonging to phase two by taking advantage of the implicit dominance relations induced by the tree structure. According to the authors, *pruning* is to be performed after $k = \left\lfloor |Y|^\delta \right\rfloor$ points have been considered in the first phase, for some $\delta \in \mathbb{R}$. In the following, we use parameter values recommended by the authors.

Algorithm 3: (**Dominated**) Check whether a point y is dominated in the subtree rooted in y'

input : $Tree$ node r, $y \in \mathbb{R}^p$
output: $True$ if y is dominated by an element in the subtree rooted in r, $False$ otherwise

```
1  /*Since a point is one of the attributes of an intermediary node of
     a KDTree, nodes can be represented by points                    */
2  /*If a node is a leaf node, its LChild and RChild pointers link to to
     a particular leafPt object                                      */
3  if r ⪰ y then
4      return True
5  if LChild(r) ≠ leafPt then
6      z ← LChild(r)
7      if u^z ⪰ y then
8          if Dominated(z, y) then
9              return True
10 if RChild(r) ≠ leafPt then
11     z ← RChild(r)
12     if u^z ⪰ y then
13         if Dominated(z, y) then
14             return True
15 return False
```

1.1.3.3 Algorithms over n-ary trees with p-dimensional leaf data (NDTrees)

Another alternative to list-based dominance filtering algorithms, is filtering by insertion into an *NDTree*, as described by Jaszkiewicz and Lust (2018), which we now introduce, following Cornu

Algorithm 4: (**Insert**) Insert point y in a KDTree

input : $Tree$ node r, coordinate l, $y \in \mathbb{R}^p$
output: Updated $Tree$

for $j \in \{1, ..., p\}$ **do**
  $u_j^r \leftarrow \max\{y_j, y_j^r\}$
if $y_l \geqslant r_l$ & $RChild(r) \neq leafPtr$ **then**
  $Insert(RChild(r), l+1 \mod p, y)$
if $y_l \geqslant r_l$ & $RChild(r) = leafPtr$ **then**
  /*Create new node in leaf position */
  $LChild(y) \leftarrow leafPtr$
  $RChild(y) \leftarrow leafPtr$
  $u^y \leftarrow y$
  $RChild(r) \leftarrow y$
if $y_l < r_l$ & $LChild(r) \neq leafPtr$ **then**
  $Insert(LChild(r), l+1 \mod p, y)$
if $y_l < r_l$ & $LChild(r) = leafPtr$ **then**
  /*Create new node in leaf position */
  $LChild(y) \leftarrow leafPtr$
  $RChild(y) \leftarrow leafPtr$
  $u^y \leftarrow y$
  $LChild(r) \leftarrow y$

(2017). An NDTree is a tree which is used as an archive to store non-dominated points. Leaf-nodes in the tree are associated with sets of points, and internal nodes are associated with points which are respectively upper and lower bounds to the elements associated with the leaf-nodes of their subtrees. An NDTree is generally used in procedures where individual *candidate* points, which may be dominated, are presented successively. Usage of NDTrees can be broken down into two main procedures.

The first procedure tests whether an incoming candidate is dominated by some point in the leaf-nodes. Because of the tree-like structure of the archive, an incoming point needs not be compared to every point in the archive. At each node, the incoming point may be dominated by the lower bound of the internal node, then it is known to be dominated, and it is rejected. It may dominate the upper bound of the internal node, and then it can eliminate that node and its subtree, and be accepted. If the candidate point weakly dominates the lower bound of a node, and is weakly dominated by the upper bound of a node, then one of three cases occurs. 1. If the node is a leaf node, then the candidate point is compared to points in the associated subset, the points it dominates are removed from it, and if it is found to be non-dominated within that set, it is added to it. 2. If the node is an internal node, the candidate is passed down to the child nodes. 3. If the candidate point is neither within the bounds of the node, nor dominated by it, nor dominating is, we stop the testing of the candidate in this subtree.

If the candidate was not found to be dominated during the previous procedure, a second procedure inserts it into the leaf node which it is closest to, for some notion of distance between a point

and a node. Since each leaf node has a maximum storage capacity, they have to be split when the capacity is exceeded by the entry of a new point. Then, upper and lower bounds of nodes in paths of the tree which end with modified leaf nodes must be updated.

We implement this method using parameter values recommended by the authors, and within the frame of the following, straightforward algorithm: for each point in Y, insert it in the NDTree. When all input points have been inserted, "harvest" the contents of the leaf-nodes of the NDTree, the union of which is $\mathcal{N}(Y)$.

1.1.4 Choosing a dominance filtering algorithm

Throughout this document, we propose to tackle optimization problems with specialized methods which, we intend to show, perform better than their generic counterparts. In doing so, we use dominance filtering as a basic algorithmic building block. In order for the performance improvements we exhibit to represent as faithfully as possible the contributions we put forth, we need to be assured that we use the best dominance filtering method available.

When $p = 2$, we will use a unidirectional algorithm described by Kung et al. (1975) in which the set Y to be filtered is first sorted in decreasing order of value of the first component. Because points are in this case of dimension 2, if an input point is dominated by a previously considered point, then it is dominated by the last point found to be non-dominated. When $p \geqslant 4$, we use the *two phase method* together with insertion into a KDTree, both proposed by Chen et al. (2012). We conducted additional preliminary experiments comparing their approach to Jaszkiewicz and Lust (2018), and found that Chen et al. (2012) performed better. When $p = 3$, the picture is less clear and we will have to experiment with both unidirectional dominance filtering algorithms and Chen et al. (2012), and always consider the one that allows the best performance for the method we are challenging.

1.2 Coupled problems and decomposition

Decomposing an optimization problem means replacing it with several simpler subproblems, and then combine their optimal solutions in some way, so as to obtain the set of optimal solutions to the original problem. This notion has frequently been investigated in the context of the optimization of *continuous complex systems* (e.g., by Gardenghi (2009); Ishibuchi et al. (2015)) in which the intuition justifying decomposition is that the sum of the computational efforts spent to solve the subsystem problems and to combine their solutions should be significantly less than the effort required to solve the whole problem. The question is then of whether we can guarantee the optimality of the combined solutions for the original problem, or at least their approximate optimality.

In the formal context of optimization problems, decomposition has two aspects: one related

to the functions to be optimized (we will call it *separability*), one related to the feasible set (we will call it *structural decomposability*). Separability has to do with how the optimization of the subsystems is reflected in that of the global system: the global evaluation of the system is obtained by aggregating the objective functions of the subsytems. Decomposition is possible when this aggregation operator satisfies a separability property. Structural decomposability has to do with how feasible solutions for the subsystems relate to feasible solutions for the original problem: feasible solutions to the global system problem should be obtained by combining feasible solutions to the subsystem problems as straightforwardly as possible.

The combination of these two aspects allows for decomposition, i.e. the computation of the desired solution by solving subproblems and combining their solutions. A complex system is, however, generally not straightforwardly decomposable. For example, it can be the case that some constraints induce such an interdependence between subsystem solutions, that the optimum for one subproblem cannot be obtained independently from those of other subproblems, or that combining solutions obtained independently in subproblems yields an infeasible solution.

Decomposition and coordination approaches, which are classical in the single objective continuous case, are usually not applicable in the multiobjective case. Informally the reason is that these approaches critically rest on the ability to define a sequence of optimizations which converges to a unique optimal value. This can be done either by finding, from a feasible solution to the problem, a direction of improvement for the evaluation of variables involved in the interaction between subsystem, or, from an infeasible solution to the problem, by a modification of the objective function that will reduce the violation of coupling constraints. In the multiobjective case, because there is no such thing as a unique optimal value, and because for each feasible solution there would be as many directions of improvement as there are objective functions, such a converging sequence cannot be defined.

1.2.1 Decomposable formulations

Let us set the stage for a generic formulation of coupled optimization problems. The subproblems into which a coupled optimization problem is to be decomposed should reflect such structural features of a complex system as: the division into technical components in the case of the design of a technical device, distinct geographic areas in a planning problem with a spatial aspect, etc. Formally, the vector of decision variables should be split into subsystem-wise subvectors. The models we study here are such that all decision variables are local to some subsystems, at least in the basic formulations. This is to restrict ourselves to the case where the existence of a coupling is the consequence of the presence of a *coupling constraint*.

Thus let $N = \{1, ..., n\}$ be the set of decision variable indices. Let a partition S of N represent the set of subsystems. A subsystem $s \in S$ is thus identified with the subset of decision variable indices involved in the constraints defining this subsystem. Thus, x, the vector of all decision variables, can be seen as $(x_1, ..., x_n)$, or $(x^s | s \in S)$, where $x^s = (x_i \mid i \in s) = (x^s_1, ..., x^s_{n_s})$ regroups the decision variables associated with subsystem s. In this framework, interactions between sub-

systems are modeled by constraints involving local variables belonging to different coupled subsystems.

Example 1.1. *Let $S = \{s_1, s_2\}$, with $|s_i| = n_i, i \in \{1,2\}$, since $N = s_1 \cup s_2$ and $s_1 \cap s_2 = \varnothing$ variable vector x can be described as $(x_1^{s_1}, ..., x_{n_1}^{s_1}, x_1^{s_2}, ..., x_{n_2}^{s_2})$, which we will rather denote by (x^{s_1}, x^{s_2}).*

Let $K \subseteq \mathcal{P}(N)$ be a set of couplings between subsystems, represented by the set of indices of the variables it couples. We assume that any coupling constraint involves variables from at least two subsystems. In the following, each coupling $k \in K$ corresponds to a unique coupling constraint indexed by k.

Definition 1.6. *Two subsystems $s, s', s \neq s'$ will be said to be* coupled *by constraint k if there exist $i \in s \cap k$ and $j \in s' \cap k$. An optimization problem will be said to be* uncoupled *if and only if $K = \varnothing$.*

Note that for some $k \in K$, the associated coupling constraint may not involve all the variables of the subsystems which are coupled by k. We denote by x^k the vector of decision variables involved in constraint k, and by x_k^s the vector of decision variables from subsystem s involved in constraint k, i.e. $x_k^s = (x_i \mid i \in s \cap k)$.

Finally, we will denote by X^s the feasible set of the suproblem associated with each $s \in S$, defined over variable subvector x^s, and for each $k \in K$, X^k will denote the feasible set, defined by coupling constraint k over variable subvector x^k. Using these notations, we can express the coupling structure in the following formulation of a coupled system optimization problem.

$$
\begin{array}{lll}
\max & f(x) = (f_1(x), ..., f_p(x)) & \\
s.t. & x^k \in X^k & \forall\, k \in K \\
 & x^s \in X^s & \forall\, s \in S
\end{array}
\qquad (P)
$$

where for all $j \in \{1, ..., p\}$, $f_j(x) = \sum_{s \in S} f_j^s(x^s)$. Assume that $|S| = m$ and $(x^s | s \in S) = (x^1, ..., x^m)$. A well-known example of a coupled problem is the block-angular structure displayed below, with, for all $s \in S$, $f^s(x) : \prod_{i \in s} X_i \to \mathbb{R}^p$, where X_i is the domain of variable x_i:

$$
\begin{array}{llllllll}
\max & f^1(x^1) & +f^2(x^2) & +\ldots & +f^m(x^m) & & & \\
s.t. & \mathbf{A}_k^1 x_k^1 & +\mathbf{A}_k^2 x_k^2 & +\ldots & +\mathbf{A}_k^m x_k^m & \leqslant & \mathbf{b}^k & \forall\, k \in K \\
 & \mathbf{C}^1 x^1 & & & & \leqslant & \mathbf{d}^1 & \\
 & & \mathbf{C}^2 x^2 & & & \leqslant & \mathbf{d}^2 & \\
 & & & \ddots & & & \vdots & \\
 & & & & \mathbf{C}^m x^m & \leqslant & \mathbf{d}^m & \\
 & x_i \in X_i & & & & & & \forall\, i \in N
\end{array}
$$

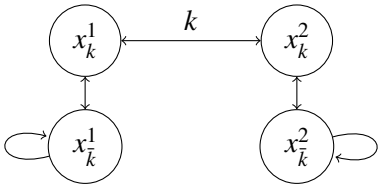


Figure 1.1: Graph representation of a coupled system described in Example 1.2

In the following, for some decision variable vector x of which a variable subvector is x^s, we denote by $x^{\bar{s}}$ the variable subvector defined by all components of x other than those of x^s. We slightly abuse notation to do this irrespective of the actual order of components in x, when it causes no confusion. In short, when we write $x \sim (x^s, x^{\bar{s}})$, we mean that $(x^s, x^{\bar{s}})$ is equal to x up to a permutation, so that we can isolate the part of it that is related to s.

Example 1.2. *Figure 1.1 provides a toy example which will illustrate the modifications later applied to obtain either restrictions or relaxations of the problem. Consider a graph $G = (V, E)$ where vertex $v \in V$ is a subset of decision variables and each edge $e = (v_i, v_j)$ represents the existence of constraints binding variables belonging to vertices v_i and $v_j \in V$. In the following graph, x_k^s for $s \in \{1, 2\}$ denotes the vector of variables of subsystem s coupled by constraint k, while $x_{\bar{k}}^s$ denotes the vector of variables of subsystem s which are not coupled by constraint k. We only label coupling constraints, i.e. constraints which link variables from different subsystems.*

Observation 1.1. *If problem (P) is uncoupled, then its feasible set X is such that*

$$X = \prod_{s \in S} X^s \tag{1.1}$$

where $\prod$ is the cartesian product of a family of sets.

Proof. Each feasible solution $x \in X$ is decomposed according to S, i.e. $x \sim (x^s \mid s \in S)$, with S a partition of N and $x^s \in X^s$ for each $s \in S$. If $K = \varnothing$, any $x = (x^s \mid s \in S)$ with $x^s \in X^s$ for all $s \in S$ is feasible, that is $x \in \prod_{s \in S} X^s$ is equivalent to $x \in X$.

□

Now that we have defined what it means to be uncoupled for a problem formulation (i.e. $K = \varnothing$) and established the link between formulation decomposability and feasible set decomposability (*i.e.* $X = \prod_{s \in S} X^s$), we will look at how optimization can take advantage of this property.

1.2.2 Uncoupled optimization

Crucial to uncoupled optimization is the fact that the objective function can be computed straightforwardly from subterms that relate to the independent subsystems. Recall that f_j is the j-th

objective function for the optimization of the global system over set X. Under the assumption that $f_j(x)$ is additively separable for all $j \in \{1, ..., p\}$, the objective function of the subproblem associated with each $s \in S$ is straightforwardly given as $f^s(x^s) = (f_1^s(x^s), ...f_p^s(x^s))$, with $f_j^s : \prod_{i \in s} X_i \to \mathbb{R}$ for $j \in \{1, ..., p\}$. Thus each subsystem is associated with a multiobjective problem $\max_{x^s \in X^s} f^s(x^s)$.

Definition 1.7. *Given $j \in \{1, ..., p\}$, $f_j : \prod_{i \in N} X_i \to \mathbb{R}^p$, and for all $s \in S$ $f_j^s : \prod_{i \in s} X_i \to \mathbb{R}^p$, f_j is said to be additively separable along S if, for all $x = (x^s \mid s \in S)$ with $x^s = (x_i \mid i \in s)$,*

$$f_j(x) = \sum_{s \in S} f_j^s(x^s)$$

For an uncoupled problem, assuming that objective functions f_j are additively separable along S for $j \in \{1, ..., p\}$, we wish to investigate the relation between the global non-dominated set $\mathcal{N}(f(X))$ and the non-dominated sets of all subsystem problems $\mathcal{N}(f^s(X^s))$, for $s \in S$. In particular, a natural question is whether the following equality is valid:

$$\mathcal{E}(X, f) \stackrel{?}{=} \prod_{s \in S} (\mathcal{E}(X^s, f^s)) \tag{1.2}$$

that is, we want to be able to obtain the efficient set of the original problem by simply combining solutions obtained by solving the subsystems problems. In Equation (C.1.2.2), decomposability of optimization is expressed in the decision space. The corresponding property, expressed in the objective space, would be:

$$\mathcal{N}(f(X)) \stackrel{?}{=} \overset{\circ}{\sum_{s \in S}} \mathcal{N}\left(f^s(X^s)\right)$$

where $\overset{\circ}{\sum}_{s \in S} Y^s$ is the Minkowski or set sum. We recall that for $p \in \mathbb{N}$, and $A, B \subseteq \mathbb{R}^p$, the Minkowski sum of A and B is defined as

$$A \oplus B := \{a + b \mid a \in A, b \in B\}$$

then for $S = \{1, ..., m\}$, we define the Minkowski sum (MS) of a finite family $(Y^s \mid s \in S)$ of finite subsets of $\mathbb{R}^p$ as

$$\overset{\circ}{\sum_{s \in S}} Y^s := \quad (...(Y^1 \oplus Y^2) \oplus ... \oplus Y^{m-1}) \oplus Y^m$$

Proposition 1.1. *If $\prod_{s \in S} X^s = X$ and for all $j \in \{1, ..., p\}$, f_j is additively separable along S, then*

$$\mathcal{N}(f(X)) \subseteq \overset{\circ}{\sum_{s \in S}} \mathcal{N}(f^s(X^s))$$

Proof. Let $f(x) \in \mathcal{N}(f(X))$, with $x = (x^s \mid s \in S)$. We prove that $f^s(x^s) \in \mathcal{N}(f^s(X^s))$ for all $s \in S$. If there is $s \in S$ and $x^s \in X^s$ such that $f^s(x^s) \notin \mathcal{N}(f^s(X^s))$, then there is $x'^s \in X^s$ such that $f^s(x'^s) \geq f^s(x^s)$. Since $X = \prod_{s\in S} X^s$ and $x'^s \in X^s$, $x^* = (x'^s, x^{\bar{s}})$ is feasible. Moreover, since for $1 \leqslant j \leqslant p$, $f_j(x) = \sum_{s\in S} f_j^s(x^s)$, we get that $f(x^*) \geq f(x)$, contradicting $f(x) \in \mathcal{N}(f(x))$ □

When $p = 1$, the converse is true, i.e. we have $\max_{x\in X} \sum_{s\in S} f^s(x^s) = \sum_{s\in S} \max_{x^s\in X^s} f^s(x^s)$. However, this is no longer the case when $p \geqslant 2$, and we have $\sum_{s\in S} \mathcal{N}(f^s(X^s)) \not\subseteq \mathcal{N}(f(X))$. In other words the combination of non-dominated solutions with cartesian product can generate dominated solutions. For a counter-example, let $S = \{1, 2\}$, $X^s = \{x^s, x'^s\}$ for $s \in S$. In the following we write $y_j'^s$ for $f_j^s(x')$. Let $x = (x^1, x^2)$, $x' = (x'^1, x'^2) \in X = X_1 \times X_2$, with the following values:

$$\begin{aligned} y^1 &= (0, 1), \qquad y^2 = (1, 0) \\ y'^1 &= (2, 0), \qquad y'^2 = (0, 2) \end{aligned}$$

we have $\mathcal{N}(f^1(X^1)) = \{y^1, y'^1\}$ and $\mathcal{N}(f^2(X^2)) = \{y^2, y'^2\}$. Therefore $y^1 + y^2 \in (\mathcal{N}(f^1(X^1)) \oplus (\mathcal{N}(f^2(X^2)))$. However $y'^1 + y'^2 \geq y^1 + y^2$, and so $y^1 + y^2 \notin \mathcal{N}(f(X))$. Thus, since $\overset{\circ}{\sum_{s\in S}} \mathcal{N}(f^s(X^s))$ may contain dominated points, only the following result holds:

Corollary 1.1. *If $X = \prod_{s\in S} X^s$ and for $j \in \{1, ..., p\}$, f_j is additively separable, then*

$$\mathcal{N}(f(X)) = \mathcal{N}(\overset{\circ}{\sum_{s\in S}} \mathcal{N}(f^s(X^s)))$$

In other words, filtering by dominance the Minkowski sum of non-dominated sets of the subproblems yields the solution of the global, uncoupled problem.

Finally, an important remark. In further applications, the process of combining solutions of subproblems, and then filtering the set of combinations, which we call a *pooling process*, intervenes at different stages of complex computations. In somes cases, we want to produce only the non-dominated points which are the result of pooling. Sometimes, even if pooling is the final stage of the computation, which can be the case for a decomposable problem, we may want to exhibit the efficient solutions associated with non-dominated points. And in some other cases, when pooling is an intermediary step in the computation, we may *need* to consider solutions rather than their values, in order to perform operations which demand that structural constraints be satisfied.

When the objective function is additively separable along S, the operation of combining two solutions is equivalent, in the objective space, to computing the sum of their values. In the decision space, this operation is a concatenation. For simplicity, we will consider that we combine objects which have two attributes: a solution vector (or matrix), and a value vector, which we will note, for object e, $sol(e)$ and $val(e)$. This way we can also make it apparent that computing the value of a combination of solutions can be done quicker than by iterating over the solution vector: we can simply compute the sum of their value attributes.

Thus, we can use the same algorithm to perform the Minkowski sum or the cartesian product of two sets of points or of solutions, and the dominance algorithm we use apply to these solution

objects irrespective of whether one wants to retrieve the solution or the value of a solution, but in mathematical notations, we will still write $\prod$ and $\mathcal{E}$ when operating explicitly over sets of solutions, and $\overset{\circ}{\sum}$ and $\mathcal{N}$ when operating explicitly over points.

1.3 The Generic Uncoupled Problem and the efficiency of decomposition

The first question we will adress empirically is that of whether decomposing an uncoupled problem, i.e. solving the subproblems independently, then pooling the solutions to the subproblems, leads to faster resolution than solving the uncoupled problem at once and as a whole. We study the case where solutions are retrieved, rather than values only.

The number of decision variables in the original problem is the sum of the number of decision variables in each subproblem. Thus, if the complexity of the resolution method applied to the whole problem is higher than linear in the number of decision variables, and if the pooling algorithm applied to the sets of efficient solutions of the subproblems is sufficiently fast, one can expect that decomposition in itself will yield a decrease in the time required to solve the problem.

We test this hypothesis by comparing the running time for solving a generic uncoupled (GUCP) MOCO problem using a generic MOCO method, with the running time for solving each subproblem using the same generic method, and pooling the obtained local non-dominated sets. Throughout this document, the generic MOCO method used is the e-constraint method. For $p = 2$ we use our own implementation of the classic algorithm by Changkong and Haimes (1983), and for $p \geqslant 3$, we use the generalized method by Tamby and Vanderpooten (2021). We will refer to them indistinctly as e-constraint methods, and the algorithm we use will be clear from the number of criteria in context.

GUCP is to be regarded as a particular case of a generic coupled GCP MOCO problem, the latter being formulated as follows :

$$\begin{aligned}
\max \quad & f(x) = \Big(\sum_{s\in S}\sum_{i\in s}\pi_1^i x_i, ..., \sum_{s\in S}\sum_{i\in s}\pi_p^i x_i\Big) & & \\
& \sum_{s\in S}\sum_{i\in s\cap k} a_i x_i & \leqslant b_k \quad & \forall\, k \in K \\
& \sum_{i\in s} c_i x_i & \leqslant d_s \quad & \forall\, s \in S \\
& x_i \in \{0,1\} & & \forall\, i \in N
\end{aligned} \tag{GCP}$$

where for all $i \in N$, $a_i, c_i, b_k, d_s \in \mathbb{R}_{\geqslant}$ and $\pi_i^j \in \mathbb{R}_{\geqslant}$ for all $j \in \{1, ..., p\}$, so that $f(x) : \{0,1\}^n \to \mathbb{N}^p$. Thus GUCP is formulated as

$$\begin{aligned} \max \quad & f(x) = \sum_{s \in S} \sum_{i \in s} \pi_i x_i & & \\ & \sum_{i \in s} c_i x_i & \leqslant d_s \quad & \forall\, s \in S \\ & x_i \in \{0,1\} & & \forall\, i \in N \end{aligned} \tag{GUCP}$$

In each of trial of the next experiment, an instance of the GUCP is generated randomly according to a set of parameters. Instances are generated as follows. Each variable is assigned to a random subsystem according to a uniform law, which results in the number of variables in each subsystems being, on average, $\frac{n}{s}$. Then for every $i \in \{1, ..., n\}$, we randomly choose $\pi_i, a_i, c_i, \in [0, 1000]$, and for each $s \in S$, $d_s = \left\lceil \frac{\sum_{i \in S} c_i}{2} \right\rceil$.

We solve GUCP instances using two methods. First, we solve the whole problem using the generic method. Second, we decompose the instance into $m := |S|$ subproblems, solve the instances using the same generic algorithm, and perform the pooling of the m non-dominated sets using a naive pooling Algorithm (NA) described by Algorithm 5, where E^s denotes the set of efficient solutions of subproblem $s \in S$, and $E^{\overrightarrow{s}}$ denotes the result of successive pooling operations up to subproblem $s \in S$.

We simply concatenate solutions subproblem after subproblem, and then we filter the set of all combinations by dominance, using the dominance filtering algorithm we found to perform best for each value of p. Note that if we wanted to retrieve only the non-dominated points of the problem, rather than the efficient solutions, we would not need to perform line 7 of Algorithm 5.

Algorithm 5: Naive sequential pooling of solutions sets (NA)

input : $(E^s \mid s \in S)$
output: $\mathcal{E}(\prod_{s \in S} E^s)$

1 $E^{\overrightarrow{1}} \leftarrow E^1$
2 **for** $s \in \{2, ..., m\}$ **do**
3 $Pooled := \varnothing$
4 **for** $e \in E^{\overrightarrow{s-1}}$ **do**
5 **for** $e' \in E^s$ **do**
6 /*$\bar{e}$ is a new solution object */
7 $sol(\bar{e}) \leftarrow (sol(e), sol(e'))$
8 $val(\bar{e}) \leftarrow val(e) + val(e')$
9 $Pooled \leftarrow Pooled \cup \{\bar{e}\}$
10 $E^{\overrightarrow{s}} \leftarrow Pooled$
11 **return** $\mathcal{E}(E^{\overrightarrow{m}})$

Each line of Table 1.1 reports parameter values and average measurements for 10 trials of the experiment. The number of criteria is denoted by p, n stands for the number of variables, $|S|$ denotes the number of subproblems, and *av.* s the average number, over trials, of variables per subproblem. $|\mathcal{E}|$ denotes the number of efficient solutions of the problem. Then, *av.*

p	n	$\|S\|$	$av\ \|s\|$	$\|\mathcal{E}\|$		$av\ \|\mathcal{E}^s\|$		$T.\ NoDec$		$T.\ Dec$		$av\ T.\mathcal{E}^s$		$T.Pool$	
2	100	2	50	134.8	[18.4]	47.25	[5.63]	10.19	[2.25]	4.71	[0.77]	2.353	[0.385]	0.005	[0.001]
2	100	3	33.3	124.8	[31.5]	23.91	[4.96]	9.92	[3.55]	3.06	[0.92]	1.011	[0.303]	0.026	[0.014]
2	100	4	25	106.1	[29.0]	14.13	[3.50]	8.81	[3.03]	2.15	[0.72]	0.505	[0.154]	0.125	[0.124]
2	120	2	60	199.4	[58.4]	67.45	[18.59]	22.74	[18.49]	8.37	[3.34]	4.181	[1.664]	0.011	[0.008]
2	120	2	60	191.3	[54.1]	65.15	[17.58]	28.64	[15.39]	11.89	[5.84]	5.940	[2.920]	0.011	[0.006]
2	120	4	30	160.3	[30.0]	19.33	[2.84]	15.69	[6.01]	3.12	[0.79]	0.693	[0.146]	0.351	[0.241]
3	30	2	15	65.9	[24.9]	18.22	[6.34]	4.52	[1.94]	1.95	[0.63]	0.971	[0.315]	0.001	[0.001]
3	30	3	10	59.9	[27.3]	8.67	[3.22]	3.96	[2.37]	1.63	[0.39]	0.544	[0.131]	0.001	[0.001]
3	30	4	7.5	66.2	[37.3]	5.52	[1.28]	3.82	[2.12]	1.80	[0.07]	0.449	[0.016]	0.002	[0.001]
3	40	2	20	218.2	[80.4]	44.75	[14.85]	22.55	[11.18]	5.03	[2.22]	2.509	[1.105]	0.008	[0.005]
3	40	2	20	202.6	[119.7]	40.50	[20.77]	19.63	[14.38]	4.37	[2.57]	2.178	[1.281]	0.008	[0.009]
3	40	4	10	180.1	[113.8]	10.05	[4.42]	15.81	[11.49]	2.49	[0.80]	0.619	[0.199]	0.013	[0.010]
4	20	2	10	70.5	[37.5]	18.65	[8.48]	6.39	[4.14]	2.55	[1.25]	1.273	[0.626]	0.001	[0.001]
4	20	3	6.7	73.1	[60.6]	6.24	[2.27]	6.13	[5.72]	1.51	[0.19]	0.503	[0.064]	0.002	[0.002]
4	20	4	5	37.9	[19.5]	3.71	[0.76]	2.50	[1.36]	1.73	[0.08]	0.432	[0.020]	0.001	[0.001]
4	25	2	12.5	230.1	[196.2]	35.80	[25.08]	36.07	[41.15]	5.50	[4.79]	2.744	[2.391]	0.007	[0.008]
4	25	2	12.5	217.9	[98.3]	29.95	[9.65]	30.92	[18.84]	3.86	[1.86]	1.927	[0.927]	0.006	[0.004]
4	25	4	6.25	84.6	[44.7]	5.38	[1.40]	8.05	[4.59]	1.98	[0.23]	0.494	[0.059]	0.002	[0.001]

Table 1.1: Running time (in seconds) for solving an uncoupled problem without, or with decomposition (average values for 10 trials).

$|\mathcal{E}^s| := (\sum_{s \in S} |\mathcal{E}^s|)/|S|$ denotes the average number of non-dominated points over subproblems, and *av. T.* $\mathcal{E}^s$ denotes the average time, over subproblems, for solving a subproblem. The *T. No Dec* column records the time spent computing the non-dominated set for the whole problem with the e-constraint method. Finally, *T.Dec* denotes the time spent solving the same problem using decomposition as previously described, and *T. Pool* is the time spent combining efficient solutions of the subproblems, and filtering it for dominance over the p original criteria.

All experiments throughout this document are performed on a computer equipped with a 2.11 GHz processor and 32,0 GB RAM. All implementations of algorithms were done in C++, except for the algorithm by Tamby and Vanderpooten (2021) which was implemented and Java an interfaced with our C++ methods. Solver IBM Ilog Cplex(TM) 12.9.0 was called to solve any integer or continuous problem, in particular within e-constraint methods.

The results of this first experiment suffice to show that decomposition yields a significant performance improvement for any value of p, even when pooling is done in the most naive way, applying dominance filtering only after having combined the solutions of all subproblems.

This performance improvement seems to scale both with the number of variables increases, and with the number of subsystems increases. The latter is explained by the fact that as we keep the number of variables fixed and further divide the problem in smaller subsystem problems, those subproblems are solved much faster, so that the time spent solving the problem with decomposition exceeds greatly the sum of the durations required to solve the subproblems. However, even if smaller subproblems have fewer efficient solutions, the time required by the dominance filtering of the set of combinations of solutions also grows very fast with the number of subproblems, as can be observed in column *T.Pool*.

For larger values of p, as computing time with the generic algorithm grows very fast, we had

to consider smaller values of n. As a consequence, the size of sets to be pooled are on average much smaller, so that filtering time is negligible, and its increase with $|S|$ is not apparent.

1.4 Conclusions

After recalling basics notions and challenges in integer multiobjective optimization, a swell as defining basic tools, such as dominance filtering algorithms, we have introduced the topic of this work: the optimization of coupled problems, which are models for complex systems.

We identified the conditions under which a problem that is composed of several subproblems can be solved by solving the subproblems independently. Our work ranges beyond the case of uncoupled problems, however our strategy for solving coupled problems requires us to be able to solves uncoupled problems as fast as possible. The reason for this requirement is that, for solving a coupled problem, we will reduce it to several uncoupled problems.

In chapter 2 we will see that multiobjective upper and lower bounds of a coupled problem can be obtained by solving uncoupled variants of that problem. Approaches developed in further chapters will attempt to reduce the main problem, or parts of it, to collections of uncoupled subproblems. In each of theses cases, the same methods for pooling sets of solutions of subproblems will be applied. In the next chapter we consider the algorithmic details of the pooling operation, and investigate ways to further increase the advantage yielded by decomposition.

Chapter 2

Computing efficiently the non-dominated subset of the Minkowski set sum [1]

Chapter Abstract

In this chapter, we study the computation of the non-dominated subset of the Minkowski sum (or set sum) of a finite family of finite sets of multidimensional, real-valued points. At first sight, this computation may be thought of as computing all combinations of points and applying dominance filtering to the resulting set of points. However, structural properties can be exploited algorithmically to perform this task more efficiently, by avoiding the comparison of some points. We propose two main approaches of doing so. First, we show that using lexicographic ordering over each set to be added to the sum, we can speed up the sorting phase required to efficiently apply dominance filtering to elements of the set sum. Second, we introduce *box*-based methods, defining dominance relations between sets of combinations of points.

Contents

2.1 The non-dominated subset of the Minkowski sum problem (NDMSP) . . . 34
2.2 Intermediary filtering . . . 35
2.2.1 Sequential Pooling . . . 35
2.2.2 Experimental Results . . . 36
2.2.3 Pairwise Pooling . . . 37
2.3 A unidirectional method for pooling . . . 40
2.3.1 Definition and proof of correction . . . 40
2.3.2 Experimental Results . . . 45
2.4 Box-based methods . . . 47
2.4.1 Box-based dominance relations . . . 47
2.4.2 Algorithms for box-based dominance filtering . . . 48
2.4.3 An algorithm for creating a family of boxes . . . 51
2.4.4 Experimental Results . . . 52
2.5 Conclusions and discussion . . . 56

[1] This chapter is based on submitted article Kerbérénès et al. (2021b)

2.1 The non-dominated subset of the Minkowski sum problem (NDMSP)

We adress the problem of computing $\mathcal{N}(\mathring{\sum}_{s\in S} Y^s)$, for $Y^s \subset \mathbb{R}^p$ for each $s \in S$, where $\mathring{\sum}$ denotes the set sum as defined in Section C.1.2.2. It is a multiobjective problem in the sense that it requires us to output the optima of a partial order relation on a set. However it is not defined as the optimization of a vector valued function of some variables, rather, each set of points in the family is given explictly. For a set $A \subseteq \mathbb{R}^p$ given explicitly, one may consider that this optimization is the mere filtering out of dominated points in A. This is true for a set that has no particular structure that we can exploit. However, the set $\mathring{\sum}_{s\in S} Y^s$ is the result of the MS operation over a family of sets, which induces some structure that can be exploited algorithmically. This operation consists in pairing and summing up points from sets of the family. Dominance filterings, pairings and summations, together with other algorithmic devices, should be organized in a way that makes computation faster than first computing the MS, and then filtering. To the best of our knowledge, this is a problem that has not yet been addressed.

Considering two finite sets $Y, Z \subset \mathbb{R}^p$, computing $\mathcal{N}(Y \oplus Z)$ requires tackling efficiently two computational challenges.

First, filtering $Y \oplus Z$ takes up to $|Y \oplus Z|^2$ dominance tests using a naive algorithm. However we can improve on this, either by taking inspiration from algorithms which transfer part of the filtering effort to a prior lexicographical ordering, and then perform a *unidirectional filtering* as in Kung et al. (1975) (an approach further denoted as *UA*), or by using tree-like data structures which reduce the number of tests necessary to determine whether a point is dominated or not, as in Chen et al. (2012). We can also impose a partial ordering on $Y \times Z$, based on the lexicographical orderings of Y and of Z, and use it to save dominance tests when filtering $Y \oplus Z$. This will be developed in Section C.2.1.3.

Second, $Y \oplus Z$ cannot be obtained in fewer than $|Y| \times |Z|$ steps. However, it may not be necessary to compute all of this set sum, since we are only interested in $\mathcal{N}(Y \oplus Z) \subseteq Y \oplus Z$. To avoid the computation of dominated sums of points, we do not test dominance between sums of points, but between *sets of sums of points*. These sets of sums of points can be represented by combinations of *boxes*, where a box delimits a subset of either Y or Z, using an upper bound and a lower bound. One comparison between sums of box bounds can eliminate many dominated sums of points. This approach will be developed in Section C.2.2.

Algorithms implementing these methods outperform the naive approach of computing the set sum of two sets, and then filtering it. Because solving NDMSP for an arbitrary family of sets can be reduced to a sequence of summing pairs of sets and filtering the results, tricks that improve the computing time $\mathcal{N}(Y \oplus Z)$ can have significant repercussions on the computing time for a family of arbitrary length. Next, Section C.2.1.1 details a simple, but crucial improvement on the most naive approach of first computing the whole of $\mathring{\sum}_{s\in S} Y^s$ and then applying dominance filtering to the result, a pair of operations we refer to as *pooling* the family of sets $(Y^s \mid s \in S)$.

2.2 Intermediary filtering

Consider a finite family $(Y^s \mid s \in S)$ of finite sets of points in $\mathbb{R}^p$. The first algorithm we consider for solving NDMSP, Algorithm 6, is a *sequential* algorithm in the sense that it considers sets Y^s to be summed one after the other. At step $s \in S$, it generates all combinations between points from Y^s and points obtained by combining points from subproblems 1 to $s-1$. The latter set is denoted by $Y^{\overrightarrow{s-1}}$. The most naive sequential algorithm generates the whole of $Y^{\overrightarrow{s-1}} \oplus Y^s$ at each step $s \in S$ of the computation.

2.2.1 Sequential Pooling

The first algorithm we consider, Algorithm 6, is a *sequential* algorithm in the sense that it considers sets Y^s to be summed one after the other. At step $s \in S$, it generates all combinations between points from Y^s and points obtained by combining points from subproblems 1 to $s-1$. The latter set is denoted by $Y^{\overrightarrow{s-1}}$. The most naive sequential algorithm, as shown by Algorithm 6 generates the whole of $Y^{\overrightarrow{s-1}} \oplus Y^s$ at each step $s \in S$ of the computation.

However, we can avoid the computation of the whole of $\overset{\circ}{\sum_{s\in S}} Y^s$ by applying dominance filtering after adding each term of the set sum. The next result guarantees that this *intermediary filtering* removes no non-dominated element of $\overset{\circ}{\sum_{s\in S}} Y^s$.

Proposition 2.1. *For any family $(Y^s \mid s \in S)$ with Y^s finite for any $s \in S$ and $S = \{1, ..., n\}$, for all $s' \leqslant n$,*

$$\mathcal{N}\left(\overset{\circ}{\sum_{1\leqslant s\leqslant s'}} Y^s\right) = \mathcal{N}\left(\mathcal{N}\left(\overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s\right) \oplus Y^{s'}\right)$$

Proof. We first prove the following lemma used twice in the proof of the proposition:

Lemma 1. Let $A = \overset{\circ}{\sum_{1\leqslant s\leqslant s'}} Y^s$ and $B = \mathcal{N}(\overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s) \oplus Y^{s'}$. Let $y \in A\backslash B$. Then there is some $y' \in B$ such that $y' \geq y$.

Proof of Lemma 1. If $y \in A\backslash B$, then $y = y^j + y^{s'}$, with $y^j \in \overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s \backslash \mathcal{N}(\overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s)$ and $y^{s'} \in Y^{s'}$. Then there is some $y'^j \in \overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s$ such that $y'^j \geq y^j$. Because Y^s is finite for any $s \in S$, $\overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s$ is also finite, so we can assume that $y'^j \in \mathcal{N}(\overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s)$. Then there is some $y' = y'^j + y^{s'}$ such that $y' \geq y$. And because $y'^j \in \mathcal{N}(\overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s)$, we have that $y' \in B$. □

Using notations of Lemma 1, let us show that $\mathcal{N}(A) \subseteq \mathcal{N}(B)$. Let $y \in \mathcal{N}(A)$, and assume, by contradiction, that $y \notin \mathcal{N}(B)$. Either $y \in B$, or $y \notin B$. If $y \in B$, then there is $y' \in B$, $y' \geq y$. Because $B \subseteq A$, we have that $y' \in A$, and then $y \notin \mathcal{N}(A)$, which contradicts our assumption. If $y \notin B$, by Lemma 1, there is some $y' \in B$ such that $y' \geq y$, which yields the same contradiction.

Now let us show that $\mathcal{N}(B) \subseteq \mathcal{N}(A)$. Let $y \in \mathcal{N}(B)$, and assume, by contradiction, that $y \notin \mathcal{N}(A)$. Because $B \subseteq A$, there must be some $y' \in A \backslash B$ such that $y' \succeq y$. This implies, because of Lemma 1, that there is some $y'' \in B$ such that $y'' \succeq y'$. By transitivity, we get $y'' \succeq y$, which means that $y \notin \mathcal{N}(B)$, in contradiction with our assumption. □

Algorithm 6 uses this fact to apply an intermediary dominance filtering after combining points from each new subset considered, so that $Y^{\overrightarrow{s}} = \mathcal{N}(Y^{\overrightarrow{s-1}} \oplus Y^s)$. In pseudo-code, this is controlled by the boolean variable $FiltEachStep$.

The filtering strength of the dominance relation is clearly impacted by the number of dimensions of the objective space: the higher the number of dimensions, the lower the likelihood that two randomly chosen points will be comparable. When the number of criteria is low, the cost of intermediary filtering is certainly worth paying, since the filtered set may be far smaller than the original set, profitably limiting the number of combinations to consider at the next step. When the number of criteria increases, the size of the filtered set tends to get closer to that of the original set. In spite of that, we observe that intermediary filtering remains quite profitable.

Algorithm 6: Sequential Pooling algorithm without (NA) or with (IF) intermediary filtering.

input : $(Y^s \mid s \in S), FiltEachStep$
output: $\mathcal{N}(\mathring{\sum}_{s \in S} Y^s)$

$Y^{\overrightarrow{1}} \leftarrow Y^1$
for $s \in \{2, ..., |S|\}$ **do**
  $Y^{\overrightarrow{s}} \leftarrow \varnothing$
  for $y \in Y^{\overrightarrow{s-1}}$ **do**
    for $y' \in Y^s$ **do**
      $Y^{\overrightarrow{s}} \leftarrow Y^{\overrightarrow{s}} \cup \{y + y'\}$
  if $FiltEachStep$ **then**
    $Y^{\overrightarrow{s}} \leftarrow \mathcal{N}(Y^{\overrightarrow{s}})$
if *not(FiltEachStep)* **then**
  $Y^{\overrightarrow{|S|}} \leftarrow \mathcal{N}(Y^{\overrightarrow{|S|}})$
return $Y^{\overrightarrow{|S|}}$

2.2.2 Experimental Results

The results of the next experiment measure the effectiveness of applying dominance filtering after each step of adding a set to the set sum, and whether it is impacted by the number of objectives. We generate families of $|S|$ sets of size $|Y^s|$ each. Every point is drawn randomly according to a uniform distribution over a p-dimensional hypercube of width 1000. We keep each newly drawn point only if it is not dominated by, and does not dominate, any previously kept point. The reason for this requirement is that, if some point $y \in Y^s$ is dominated by $y' \in Y^s$, then any sum of points

involving point y would be dominated by the same sum where y is replaced by y'. Here, the specific computational challenge we address is the elimination of dominated combinations of points in $\mathring{\sum}_{s\in S} Y^s$, rather than that of points already dominated in some Y^s, which would necessarily be eliminated more efficiently by prior filterings of sets Y^s, for $s \in S$. We generate 20 instances for each parameter set, and compare the computation time of the naive sequential pooling algorithm (NA) when dominance filtering is applied only after having computed the MS entirely, and when it is applied after each addition of an individual set to the set sum (IF).

Algorithm 7: Unidirectional Dominance Filtering for $p = 2$

input : $Y \subseteq \mathbb{R}^p$
output: $\mathcal{N}(Y)$
$Sort(Y, \geqslant_{Lex})$
/*The for loop iterates according to $\geqslant_{Lex}$ order */
for $y \in Y$ **do**
  /*Assume $\widetilde{Y} = \{y^1, ..., y^m\}$ */
  if $y^m \not\geq y$ **then**
    $\widetilde{Y} \leftarrow Y \cup \{y\}$
return $\widetilde{Y}$

Table 2.1 reports these two values, as well as the *time gain* (TG), defined as $100\frac{T.NA-T.IF}{T.NA}$, where $T.NA$ denotes the computing time when filtering once at the end, and $T.IF$ when filtering at each step. *SR* denotes, in percentage, the reduction in size from the MS to its non-dominated subset. For example, in a trial with $p = 2$, $|S| = 3$, and $|Y^s| = 100$, the MS has a size $|\mathring{\sum}_{s\in S} Y^s| = 10^6$, whereas the average size of the non-dominated set, denoted $|ND|$, is 561.9, corresponding to a size reduction of 99.94%. Values between squared brackets correspond to standard deviations.

Table 2.2 reports step-wise measurements for the IF version, on the same instances. For $s \in S$, *IF Time Step s* denotes dominance filtering time at step s (so that, e.g., for 3 sets, there are 2 such steps, when filtering $Y^1 \oplus Y^2$ and then filtering $\overrightarrow{Y^2} \oplus Y^3$). For $s \in S$, *IF Size Reduction Step s* denotes the reduction in size from $Y^{\overrightarrow{s}}$ to $\mathcal{N}(Y^{\overrightarrow{s}})$. As regards to these measurements, experimental results suggest that the computational burden rests heavily on later stages of the computation. This is apparent in the fact that step wise size reduction increases from one step to the next. This imbalance increases significantly with p: for 5 objectives and 4 sets, step 2 takes approximately 29 times longer than step 1, and step 3 approximately 9 times longer than step 2. Meanwhile, for 4 objectives and 4 sets, step 2 takes approximately 16 times longer than step 1, and step 3 approximately 5 times longer than step 2.

2.2.3 Pairwise Pooling

As an alternative to the sequential structure of the sequential pooling algorithm with intermediary filtering (SqIF), we investigate whether solving NDMSP by summing and filtering separately pairs of sets, and then iterating the summing and filtering on the results of this operation, could lead to better results. A similar line of inquiry was developed by in the context of a particular dynamic

p	$\|S\|$	$\|Y^s\|$	$\|ND\|$		SR (%)		$NA\ Time$		IF			
									$Time$		TG (%)	
2	3	100	552.60	[125.96]	99.95	[0.01]	3.17	[0.08]	0.06	[0.01]	98.22	[0.31]
3	3	100	1523.82	[295.30]	99.85	[0.03]	4.28	[0.62]	0.41	[0.11]	90.54	[2.16]
4	3	100	5265.10	[1038.13]	99.47	[0.10]	7.27	[0.93]	1.17	[0.28]	83.76	[3.96]
5	3	100	14798.60	[3610.59]	98.52	[0.36]	12.55	[1.70]	3.48	[0.86]	72.42	[4.87]
2	3	125	654.18	[153.81]	99.97	[0.01]	10.11	[0.24]	0.16	[0.02]	98.42	[0.23]
3	3	125	1821.82	[367.57]	99.91	[0.02]	14.53	[1.36]	0.74	[0.13]	94.92	[0.92]
4	3	125	7752.73	[1595.51]	99.60	[0.08]	21.40	[2.16]	2.40	[0.59]	88.81	[2.42]
5	3	125	21380.10	[4460.15]	98.91	[0.23]	32.81	[2.92]	6.73	[1.45]	79.55	[3.45]
2	4	60	544.08	[150.59]	100.00	[0.00]	165.89	[26.25]	0.18	[0.06]	99.89	[0.03]
3	4	60	2486.45	[584.10]	99.98	[0.01]	277.19	[11.55]	1.14	[0.25]	99.59	[0.09]
4	4	60	9846.20	[2872.89]	99.92	[0.02]	309.14	[13.56]	3.82	[1.13]	98.77	[0.33]
5	4	60	35223.10	[9897.77]	99.73	[0.08]	435.65	[63.42]	15.65	[4.92]	96.39	[1.06]

Table 2.1: Global performance improvement due to filtering at each step, as a function of the number of objectives, the number of sets and the number of points in each set. Average values for 20 trials.

p	$\|S\|$	$\|Y^s\|$	$IF\ Time$						$IF\ Size\ Reduction$					
			Step 1		Step 2		Step 3		Step 0		Step 1		Step 2	
2	3	100	0.012	[0.002]	0.037	[0.009]			97.178	[0.540]	98.020	[0.414]		
3	3	100	0.045	[0.009]	0.351	[0.101]			94.604	[0.914]	97.164	[0.349]		
4	3	100	0.078	[0.013]	1.065	[0.271]			89.014	[2.025]	95.177	[0.674]		
5	3	100	0.136	[0.026]	3.219	[0.817]			80.228	[3.030]	92.563	[1.166]		
2	3	125	0.028	[0.002]	0.088	[0.018]			97.814	[0.358]	98.469	[0.247]		
3	3	125	0.083	[0.013]	0.573	[0.124]			95.902	[0.619]	97.724	[0.284]		
4	3	125	0.143	[0.018]	2.053	[0.546]			90.302	[1.726]	95.898	[0.495]		
5	3	125	0.233	[0.027]	6.020	[1.370]			83.333	[2.460]	93.432	[0.915]		
2	4	60	0.007	[0.003]	0.034	[0.012]	0.071	[0.026]	95.215	[1.251]	96.646	[0.574]	97.341	[0.333]
3	4	60	0.025	[0.005]	0.194	[0.048]	0.704	[0.164]	90.114	[1.812]	94.843	[0.759]	96.210	[0.418]
4	4	60	0.032	[0.008]	0.500	[0.162]	2.673	[0.849]	82.689	[3.257]	91.881	[1.127]	94.589	[0.871]
5	4	60	0.045	[0.009]	1.312	[0.297]	11.864	[3.998]	72.479	[4.155]	87.243	[1.571]	92.334	[1.193]

Table 2.2: Step-wise increase in performance due to filtering at each step, as a function of the number of objectives, the number of sets and the number of points in each set. Average values for 20 trials.

programming algorithm by Stiglmayr et al. (2014), who named this alternative pooling scheme "cascadic". The implementation of this approach is described by Algorithm 8.

We iterate through the family of sets using an iterator incremented by two at each step. This iterator defines a pair of sets, of which we compute the MS, before to filter the latter by dominance. The result is appended to the family of sets that will be subjected to the same operation at the next step of the algorithm. If, at any step of the algorithm, the current family of sets to be pooled is of odd length, then there is one set remaining after iterating through the family with the twin iterator. This remaining set is added to the next family of sets to be pooled. If the current family of sets to be pooled has only one element, it is the result. Intuitively, pairwise pooling should perform very similarly to the SqIF for $|S| = 2, 3$, since the sequence of pooling operations will be the same in both approaches.

Algorithm 8: Pairwise Pooling

input : $(Y^s \mid s \in S)$
output: $\mathcal{N}(\mathring{\sum}_{s\in S} Y^s)$
$Q \leftarrow \{Y^1, Y^2, ..., Y^{|S|}\}$
$i \leftarrow 1$
while $|Q| > 1$ **do**
  $Q' \leftarrow \varnothing$
  while $i + 1 < |Q|$ **do**
    $Y' \leftarrow \mathcal{N}(Y^i \oplus Y^{i+1})$
    $Q' \leftarrow Q' \cup \{Y'\}$
    $i \leftarrow i + 2$
  if $i = |Q|$ **then**
    $Q' \leftarrow Q' \cup \{Y^i\}$
  $Q \leftarrow Q'$
return $Y' \in Q$

We test this approach against the SqIF. Instances are generated in the same way as in the previous experiment. Results are provided in Table 2.3. In this experiment, variability was high, and only for the case of $p = 2$, $|S| = 3$ and $|Y^s| = 100$ did we find average time gain relative to *SqIF* which remained positive within one standard deviation. It was not, however, part of any apparent trend. In the context of their application problem, Stiglmayr et al. (2014) had found pairwise, or *cascadic* pooling to perform worse than the sequential alternative. For these reasons, we doubt that pairwise pooling can constitute a reliable alternative to sequential pooling, and we do not consider it anymore.

In this section, we have seen that when computing $\mathcal{N}(\mathring{\sum}_{s\in S} Y^s)$ for $|S| > 2$, the enumeration of many elements of $\mathring{\sum}_{s\in S} Y^s$ can be avoided by filtering after each step. But it still requires us to compute all of $\overrightarrow{Y^{s-1}} \times Y^s$ at each step s, which cannot be done in less than $O(|\overrightarrow{Y^{s-1}}| \times |Y^s|)$ steps. However because we are aiming at $\mathcal{N}(\overrightarrow{Y^{s-1}} \oplus Y^s) \subseteq \overrightarrow{Y^{s-1}} \oplus Y^s$, we can hope to avoid enumerating the combinations of points that we can prove in advance will lead to dominated points at step s. We present two types of approaches to achieve that end: a *unidirectional pooling method*, and *box-based methods*.

p	$\|S\|$	$\|Y^s\|$	$\|ND\|$	$SqIF$	$PW\ Pool$	
					$Time$	$Time\ Gain\ (\%)$
2	2	400	1133.3 [275.1]	1.376 [0.172]	1.373 [0.188]	0.292 [4.254]
3	2	400	2055.2 [361.7]	2.86 [0.645]	2.794 [0.574]	1.874 [3.904]
4	2	400	4712.4 [536.8]	10.879 [2.85]	10.809 [2.691]	-0.785 [15.921]
2	3	100	550.7 [120.5]	0.248 [0.058]	0.244 [0.053]	1.519 [3.647]
3	3	100	1540.5 [264.4]	1.371 [0.359]	1.357 [0.354]	0.91 [2.132]
4	3	100	5859.8 [1005.5]	14.218 [5.493]	14.02 [5.24]	0.824 [3.511]
2	4	40	329.8 [77.6]	0.078 [0.014]	0.082 [0.018]	-5.065 [14.5]
3	4	40	1589 [446.6]	1.431 [0.681]	1.381 [0.634]	1.897 [10.61]
4	4	40	6000.9 [1360.4]	15.401 [6.486]	15.157 [6.277]	1.308 [4.646]
2	5	20	280.4 [85.9]	0.042 [0.011]	0.043 [0.01]	-2.726 [11.01]
3	5	20	1384.1 [216.1]	0.971 [0.302]	0.965 [0.287]	0.247 [2.989]
4	5	20	5295.7 [1155.9]	12.898 [5.379]	12.776 [5.255]	0.312 [7.079]

Table 2.3: Time Gain for Pairwise Pooling relative to Sequential Pooling with filtering at each step. Average values for 20 trials.

2.3 A unidirectional method for pooling

2.3.1 Definition and proof of correction

The application of a unidirectional dominance filtering algorithm (UF) requires that the input elements $y \in Y$ be presented in a *dominance preserving order*, and one such order is the lexicographic order $\succeq_{Lex}$. Given two sets of points $Y, Z \subseteq \mathbb{R}^p$, in order to obtain $\mathcal{N}(Y \oplus Z)$, we could compute all of $Y \oplus Z$ and sort it according to $\succeq_{Lex}$, obviously producing a sequence over $Y \oplus Z$, which would allow the use of a UF algorithm. However, this section presents an alternative procedure to generate such a sequence over $Y \oplus Z$ without requiring the sorting and prior computation of $Y \oplus Z$. Rather, it requires only that Y and Z be sorted in a dominance preserving order.

Assume Y and Z are both sorted lexicographically, and indexed according to their rank in each respective lexicographic order. We define a structure in which $y^i + z^j \in Y \oplus Z$ can either have two children, $y^{i+1} + z^j$ and $y^i + z^{j+1}$, if $i + 1 \leqslant |Y|$ and $j + 1 \leqslant |Z|$ respectively, or only one child if only one of these conditions is met, or no child if none of the conditions is met. Similarly, any element $y^i + z^j$ has at most two parents: $y^{i-1} + z^j$ and $y^i + z^{j-1}$, if $i - 1 > 0$ and $j + 1 > 0$ respectively.

This *parenthood* relation defines a lattice over $Y \oplus Z$ which is a subset of $\succeq_{Lex}$ over $Y \oplus Z$, as illustrated in Figure C.2. We use a table of size $|Y| \times |Z|$, denoted by T, to record, at entry (i, j), the number of parents of $y^i + z^j \in Y \oplus Z$ which have not yet been tested for insertion at any iteration of the procedure. By definition, at the beginning of the procedure, the entry for the top of the lattice reads 0, entries for elements of the leftmost and rightmost paths in the lattice read 1, and all others read 2.

Example 2.1. *Let* $Y = \{(6,3),(3,5),(2,7),(1,10)\}$*, and* $Z = \{(7,4),(6,5),(5,6),(2,9)\}$*, sorted by decreasing lexicographic order and indexed accordingly. Let* T *stand for the matrix representation of the parenthood relation. Figure C.3 exhibits table* T *and Figure C.2 the parenthood*

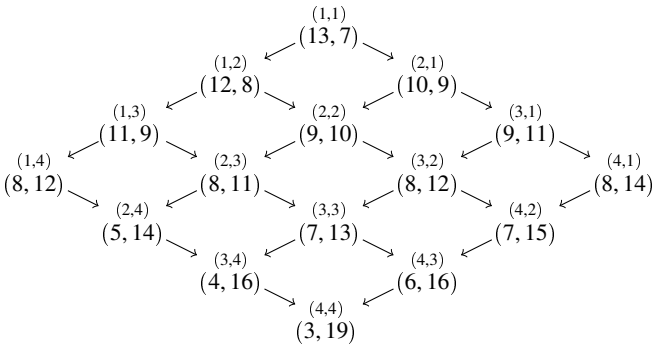


Figure 2.1: Lattice-representation of the parenthood relation over $Y \oplus Z$ in Example C.1

$T =$

	1	2	3	4
1	0	1	1	1
2	1	2	2	2
3	1	2	2	2
4	1	2	2	2

Figure 2.2: Table representation of the parenthood relation over $Y \oplus Z$ in Example C.1

relationship as a lattice.

Let $P \subseteq \mathcal{N}(Y \oplus Z)$ denote the set $\mathcal{N}(Y \oplus Z)$ being constructed. Let H denote the subset of elements of $Y \oplus Z$ whose associated entry in T is 0. H is structured as a heap ordered by decreasing lexicographic order. It will be used to determine the next point to be tested for insertion, so as to ensure that the sequence of insertions is the $\succeq_{Lex}$ order over $Y \oplus Z$.

Algorithm 9 (UPool) describes the unidirectional pooling process. The general idea is to guarantee that sum points are tested in decreasing lexicographic order. First, Y^1 is sorted. As in Algorithm 6, each iteration of the procedure adds set Y^s to set sum $\overrightarrow{Y^{s-1}}$. In the case of UPool, this set sum is also being kept sorted, so we do not need to sort $\overrightarrow{Y^{s-1}}$ before computing $\mathcal{N}(\overrightarrow{Y^{s-1}} \oplus Y^s)$.

At initialization, we know that point $y^1 + z^1$ belongs to $\mathcal{N}(\overrightarrow{Y^{s-1}} \oplus Y^s)$, and can be inserted into P (line 6). The next two largest elements are its children, $y^1 + z^2$ and $y^2 + z^1$, which now have no untested parents, and are inserted into heap H in lexicographic order (line 7). Parenthood table T is initialized accordingly (lines 8 to 15). We continue inserting elements h which appear at the root of H into P and update H and T until $H = \varnothing$.

We check whether h is dominated in P (line 18), using a generic procedure $notDominatedIn(y, A)$ which determines whether point y is non-dominated by an element of set A. In practice, if $p = 2$, it is sufficient to compare the incoming point with the last inserted point, as in Kung et al. (1975). If $p \geqslant 4$, previously accepted points are stored into a KDTree and we check whether the incoming point is dominated in the KDTree, adapting the approach by Chen et al. (2012). If $p = 3$, we may use either a generalized unidirectional method of dominance testing with heuristics such as

move-to-front and insertion at the front of P, or we may use our adaptation of Chen et al. (2012).

If h is not dominated, we add it to P (line 19). We then have to update H and T, as described in Algorithm 10. We remove h from H (line 2) and for each child of h, if it exists (line 3 or 7), we decrement its entry in T (lines 4 and 8). If any of these has its two parents already tested (lines 5 or 9), we insert it into H in the correct place according to $\succeq_{Lex}$ (line 6 or 10).

Algorithm 9: Unidirectional Pooling Algorithm (**UPool**)

input : $(Y^s \mid s \in S)$
output: $\mathcal{N}(\mathring{\sum}_{s\in S} Y^s)$

1 $sort(Y^1, \succeq_{Lex})$
2 $Y^{\overrightarrow{1}} \leftarrow Y^1$
3 **for** $s \in \{2, ..., |S|\}$ **do**
4 $\quad Y \leftarrow Y^{\overrightarrow{s-1}}$; $Z \leftarrow Y^s$
5 $\quad sort(Z, \succeq_{Lex})$
6 $\quad P \leftarrow \{y^1 + z^1\}$
7 $\quad H \leftarrow \{y^1 + z^2, y^2 + z^1\}$ /*inserted according to $\succeq_{Lex}$ */
8 $\quad T(1,1) \leftarrow 0$; $T(1,2) \leftarrow 0$; $T(2,1) \leftarrow 0$
9 $\quad$ **for** $i \in \{3, ..., |Y|\}$ **do**
10 $\quad\quad T(i,1) \leftarrow 1$
11 $\quad$ **for** $j \in \{3, ..., |Z|\}$ **do**
12 $\quad\quad T(1,j) \leftarrow 1$
13 $\quad$ **for** $i \in \{2, ..., |Y|\}$ **do**
14 $\quad\quad$ **for** $j \in \{2, ..., |Z|$ **do**
15 $\quad\quad\quad T(i,j) \leftarrow 2$
16 $\quad$ **while** $H \neq \varnothing$ **do**
17 $\quad\quad h \leftarrow root(H)$
18 $\quad\quad$ **if** *notDominatedIn(h, P)* **then**
19 $\quad\quad\quad P \leftarrow P \cup \{h\}$ /*For $p = 2$, insertion is at the back. For $p \geqslant 3$, it may be at the front or at the back depending on the chosen heuristic */
20 $\quad\quad update(h, H, T, Y, Z)$
21 $\quad Y^{\overrightarrow{s}} \leftarrow P$
22 **return** $Y^{\overrightarrow{|S|}}$

Example 2.2 shows an execution of Algorithm 9.

Example 2.2. *In the following, using the same sets Y and Z and notations as in Example C.1, we develop an execution of the algorithm for illustrative purposes. Let H represent the heap, and P represent the set $\mathcal{N}(Y \oplus Z)$ being constructed.*

0. We begin with $P = \{\overset{(1,1)}{(13,7)}\}$ and $H = \{\overset{(1,2)}{(12,8)}, \overset{(2,1)}{(10,9)}\}$. We also have $T(1,1) = 0$, and since its children are in H, $T(1,2) = 0$ and $T(2,1) = 0$.

1. $(12,8)$ is at the top of the heap and is tested for insertion in P. As it is not dominated by any

Algorithm 10: Update heap H and tree T (**update**)

```
input : h, H, T, Y, Z
/*Assume that h = y^i + z^j                                   */
H ← H\{h}
if i + 1 ⩽ |Y| then
    T(i + 1, j) ← T(i + 1, j) − 1
    if T(i + 1, j) = 0 then
        H ← H ∪ {y^{i+1} + z^j}
if j + 1 ⩽ |Z| then
    T(i, j + 1) ← T(i, j + 1) − 1
    if T(i, j + 1) = 0 then
        H ← H ∪ {y^i + z^{j+1}}
```

element of P, $P \leftarrow P \cup \{(13,7),(12,8)\}$. We decrement entries of T associated with both children of $(1,2)$, i.e. $T(2,2) \leftarrow 2-1=1$ and $T(1,3) \leftarrow 1-1=0$. Thus only $\overset{(1,3)}{(11,9)}$ is inserted into H, while $\overset{(1,2)}{(12,8)}$ is removed.

2. $H = \{\overset{(1,3)}{(11,9)}, \overset{(2,1)}{(10,9)}\}$. $(11,9)$ *is at the top of the heap and is tested for insertion in P. It is not dominated in P, and thus* $P \leftarrow \{(13,7),(12,8),(11,9)\}$. $T(2,3) \leftarrow 2-1$ *and* $T(1,4) \leftarrow 1-1=0$.

3. $H = \{\overset{(2,1)}{(10,9)}, \overset{(1,4)}{(8,12)}\}$. $(10,9)$ *is at the top of the heap and is tested for insertion in P. It is dominated in P by* $(11,9)$*, and thus* $P = \{(13,7),(12,8),(11,9)\}$. $T(3,1) \leftarrow 1-1=0$*, and* $T(2,2)=1-1=0$.

4. $H = \{\overset{(3,1)}{(9,11)}, \overset{(2,2)}{(9,10)}, \overset{(1,4)}{(8,12)}\}$. $(9,11)$ *is not dominated in P and thus* $P \leftarrow \{(13,7),(12,8),(11,9),(9,11)\}$. $T(4,1) \leftarrow 1-1=0$, $T(3,2) \leftarrow 2-1=1$.

5. $H = \{\overset{(2,2)}{(9,10)}, \overset{(4,1)}{(8,14)}, \overset{(1,4)}{(8,12)}\}$. $(9,10)$ *is dominated by* $(9,11)$ *and thus P is unchanged.* $T(3,2) \leftarrow 1-1=0, T(2,3) \leftarrow 1-1=0$.

6. $H = \{\overset{(4,1)}{(8,14)}, \overset{(1,4)}{(8,12)}, \overset{(3,2)}{(8,12)}, \overset{(2,3)}{(8,11)}\}$. $(8,14)$ *is not dominated in P, thus* $P \leftarrow P \cup \{(8,14)\}$.
   $(4,2)$ *is the only child of* $(4,1)$*, and thus* $T(4,2) \leftarrow 2-1=1$.

7. $H = \{\overset{(1,4)}{(8,12)}, \overset{(3,2)}{(8,12)}, \overset{(2,3)}{(8,11)}\}$. $(8,12)$ *is dominated by* $(8,14)$*, so that P remains unchanged.* $(2,4)$ *is the only child of* $(1,4)$*, thus* $T(2,4) \leftarrow 2-1=1$

8. $H = \{\overset{(3,2)}{(8,12)}, \overset{(2,3)}{(8,11)}\}$. $(8,12)$ *is dominated by* $(8,14)$*, so that P remains unchanged.* $T(4,2) \leftarrow 1-1=0$, $T(3,3) \leftarrow 2-1=1$

9. $H = \{\overset{(2,3)}{(8,11)}, \overset{(4,2)}{(7,15)}\}$. $(8,11)$ *is dominated by* $(8,14)$*, so that P remains unchanged.* $T(3,3) \leftarrow 1-1=0$, $T(2,4) \leftarrow 1-1=0$.

10. $H = \{\overset{(4,2)}{(7,15)}, \overset{(3,3)}{(7,13)}, \overset{(2,4)}{(5,14)}\}$. $(7,15)$ *is not dominated in P, so that* $P \leftarrow P \cup \{(7,15)\}$. $T(4,3) \leftarrow 2-1=1$.

11. $H = \{\overset{(3,3)}{(7,13)}, \overset{(2,4)}{(5,14)}\}$. $(7,13)$ *is dominated by* $(7,15)$*, so that P remains unchanged.* $T(4,3) \leftarrow 1-1=0$, $T(3,4) \leftarrow 2-1=1$

12. $H = \{\overset{(4,3)}{(6,16)}, \overset{(2,4)}{(5,14)}\}$. $(6,16)$ *is not dominated in P, so that* $P \leftarrow P \cup \{(6,16)\}$. $T(4,4) \leftarrow 2-1=1$.

13. $H = \{\overset{(2,4)}{(5,14)}\}$. $(5,14)$ *is dominated by* $(6,16)$*, so that P is unchanged.* $T(3,4) \leftarrow 1-1=0$.

14. $H = \{\overset{(3,4)}{(4,16)}\}$. $(4,16)$ *is dominated by* $(6,16)$*, so that P is unchanged.* $T(4,4) \leftarrow 1-1=0$.

15. $H = \{\overset{(4,4)}{(3,19)}\}$. $(3,19)$ *is not dominated in P so that* $P \leftarrow \{(13,7),(12,8),(9,11),(8,14),(7,15),(6,16),(3,19)\}$

16. $H = \varnothing$*, end.*

Next, we prove that Algorithm 9 yields the correct result, by showing that it fulfills the conditions for being a unidirectional dominance filtering algorithm.

Proposition 2.2. *Let Y and Z be two subsets of points. Algorithm 9 generates* $\mathcal{N}(Y \oplus Z)$.

Proof. Algorithm 9 is a unidirectional dominance filtering algorithm. Therefore, correctness is established if all elements of $Y \oplus Z$ are tested by decreasing lexicographic order $\succeq_{Lex}$. For this purpose, we need to show that, (1) except for $y^1 + z^1$, all elements of $Y \oplus Z$ are inserted into H at some iteration, and will thus be tested, and (2) the root of H at any iteration is lexicographically larger than all untested points.

For (1) let us assume by contradiction that $g \in Y \oplus Z$ never appeared in H. This means that at least one of the parents of g has not been tested either, which means it did not appear in H. Reiterating this, we exhibit a chain of elements that never appeared in H, leading to the root element of the parenthood relation. This means that either $y^1 + z^2$ or $y^2 + z^1$ was never in H, which contradicts line 7 of Algorithm 9.

For (2), let us assume by contradiction that, at some iteration of the while loop (lines 15-20), $h \in Y \oplus Z$ is the root of H as in line 17, while there exists an untested element $h' \in Y \oplus Z$ such that $h' \succeq_{Lex} h$. By line 7, we know that this may not occur at the first iteration. Therefore, we consider the earliest iteration where such a point h' might exist. Clearly h' is an untested point that does not belong to H, since H is sorted according to $\succeq_{Lex}$. However, there must exist $h'' \in H$ which is an ancestor of h'. If it were not the case, h' could never itself be brought into H, because h'' would not be tested, implying that none of its descendants, among which h', would never join H either. This would contradict (1). H being sorted lexicographically, we have $h \succeq_{Lex} h''$. Because h'' is an ancestor of h', we have $h'' \succeq_{Lex} h'$, which, by transitivity of $\succeq_{Lex}$, implies that $h \succeq_{Lex} h'$, contradicting the assumption. □

We now compare IF and UPool from a computational complexity point of view. For this purpose, we consider both algorithms at a given iteration of the global for loop, when combining two sets Y and Z. Assume that $|Y| = m$, $|Z| = n$, with $m \geqslant n$. IF and UPool differ on the prior sorting. IF requires the sorting of $Y \oplus Z$, performed in $O(mn \log mn) = O(mn \log m)$ steps. UPool requires the sorting of Y and of Z, performed in $O(m \log m + n \log n) = O(m \log m)$ steps, followed by the insertion of the mn elements into the intermediary heap structure H. By Proposition 2.3, we prove that the size of H is always less than n. Therefore, this second operation requires, in total, $O(mn \log n)$ steps.

Proposition 2.3. *Given two sets Y and Z to be combined by Algorithm 9, the heap H used in this algorithm never contains more than* $\min\{|Y|, |Z|\}$ *elements.*

Proof. Let $Y = \{y^1, ..., y^m\}$ and $Z = \{z^1, ..., z^n\}$, with $m \geqslant n$. Assume by contradiction that $|H| > n$. Then, H must contain at least two elements of the form $y^i + z^j$ and $y^k + z^j$. Assume, wlog, that $i < k$, meaning that $y^i + z^j$ is an ancestor of $y^k + z^j$ in the parenthood lattice. An element is introduced in H when its entry in T is decremented to 0. The entry of an element in T is decremented only when both its father nodes, and thus all its ancestor nodes, have been removed from H. Thus, if $y^k + z^j \in H$, then $y^i + z^j \notin H$, contradicting the assumption. □

Both algorithms require filtering the mn elements of $Y \oplus Z$, which involves mn operations in the biobjective case, and at least $O(mn \log mn) = O(mn \log m)$ operations for more than two objectives. In the latter case, this term will in all likelihood dominate the one associated with sorting. Thus we cannot guarantee that UPool will perform better than IF, but we showed that the structure of NDMSP can be taken advantage of to shorten significantly the sorting phase. In the biobjective case, sorting corresponds to the dominant term, with an overall complexity of $O(mn \log m)$ for IF, versus $O(mn \log n)$ for UPool. Therefore, we can expect UPool to perform better than IF in that case.

2.3.2 Experimental Results

We generate families $(Y^s \mid s \in S)$ in the same way as in the experiment about intermediary filtering (cf. Section C.2.1.2), and we measure time gain (TG) yielded by UPool relative to IF, using the dominance filtering algorithm which allows IF to perform best. It can be either UF with *move-to-front* heuristic and insertion of new non-dominated elements at the front, denoted by UF, or the two-phase algorithm by Chen et al. (2012), denoted by CH.

For $p = 2$ and $p = 3$, we found that IF performs best using UF for intermediary filtering (cf. bold-titled column), and thus we consider TG when testing and inserting elements into P using UF as well. We find significant TG, up to $81,65$ % for $p = 2$ with a monotonous increase as $|Y^s|$ increases, and low variability. For $p = 3$, we find the same monotonous increase, but only 49.33% TG at best. In both cases we observe a very slight increase of TG as $|S|$ increases, suggesting the performance of the algorithm scales well with the size of the sets being handled. These results are reported in Table 2.4.

p	$\|S\|$	$\|Y^s\|$	**IF UF**	**UPool UF**		*IF CH*	*UPool CH*	
			Time	*Time*	*TG/UF(%)*	*Time*	*Time*	*TG/CH(%)*
2	2	200	0.197 [0.05]	0.072 [0.02]	63.322 [8.87]	0.383 [0.11]	0.086 [0.02]	77.668 [5.67]
2	2	400	0.466 [0.12]	0.132 [0.03]	71.675 [6.70]	0.717 [0.14]	0.171 [0.04]	76.100 [6.61]
2	2	600	0.923 [0.06]	0.230 [0.02]	75.085 [1.27]	1.305 [0.25]	0.334 [0.02]	74.372 [4.97]
2	3	200	0.314 [0.06]	0.099 [0.02]	68.635 [4.21]	0.805 [0.27]	0.121 [0.02]	85.014 [4.42]
2	3	400	1.294 [0.30]	0.346 [0.07]	73.268 [2.38]	3.171 [0.83]	0.485 [0.11]	84.712 [3.23]
2	3	600	4.357 [1.59]	0.861 [0.17]	80.247 [6.30]	7.467 [1.64]	1.342 [0.22]	82.030 [3.68]
2	4	200	0.952 [0.38]	0.241 [0.05]	74.690 [4.99]	2.764 [1.00]	0.316 [0.08]	88.573 [2.66]
2	4	400	5.178 [2.16]	1.056 [0.39]	79.604 [4.17]	13.173 [6.09]	1.580 [0.64]	88.008 [2.49]
2	4	600	11.534 [3.19]	2.117 [0.62]	81.648 [2.27]	30.086 [14.25]	3.518 [1.21]	88.308 [2.91]
3	2	200	0.158 [0.03]	0.115 [0.02]	26.872 [7.92]	0.192 [0.04]	0.387 [0.06]	−101.879 [39.44]
3	2	300	0.339 [0.04]	0.220 [0.05]	34.976 [7.04]	0.400 [0.07]	0.859 [0.13]	−114.893 [39.70]
3	2	400	0.668 [0.09]	0.393 [0.08]	41.138 [6.90]	0.715 [0.11]	1.774 [0.42]	−148.161 [53.44]
3	3	200	1.366 [0.51]	0.906 [0.39]	33.664 [5.96]	1.736 [0.47]	3.393 [1.03]	−95.447 [41.16]
3	3	300	2.902 [0.85]	1.546 [0.54]	46.716 [4.69]	3.731 [1.04]	7.536 [2.18]	−101.997 [35.78]
3	3	400	5.886 [1.77]	3.013 [1.23]	48.815 [6.19]	7.048 [1.48]	14.754 [4.72]	−109.355 [43.69]
3	4	200	7.049 [3.11]	4.522 [2.43]	35.852 [4.64]	8.131 [2.51]	16.123 [6.80]	−98.289 [39.88]
3	4	300	15.396 [4.78]	8.342 [3.68]	45.813 [5.15]	17.755 [4.01]	37.823 [10.29]	−113.029 [27.41]
3	4	400	29.178 [12.22]	14.785 [9.27]	49.328 [6.68]	34.559 [10.53]	67.384 [25.11]	−94.984 [30.92]

Table 2.4: Computing time for the sequential method with intermediary filtering, the UPool method, and relative time gain as a function of $|Y^s|$ and $|S|$, for $p = 2, 3$. Average values for 20 trials.

p	$\|S\|$	$\|Y^s\|$	*IF UF*	*UPool UF*		**IF CH**	**UPool CH**	
			Time	*Time*	*TG/UF(%)*	*Time*	*Time*	*TG/CH(%)*
4	2	100	0.089 [0.02]	0.109 [0.02]	−22.409 [11.36]	0.082 [0.02]	0.097 [0.01]	−17.112 [18.30]
4	2	150	0.249 [0.06]	0.268 [0.07]	−7.712 [8.90]	0.184 [0.03]	0.215 [0.02]	−17.053 [22.79]
4	2	200	0.436 [0.09]	0.440 [0.09]	−0.976 [7.68]	0.320 [0.05]	0.368 [0.05]	−15.029 [27.05]
4	3	100	2.247 [0.94]	2.034 [0.88]	9.484 [5.21]	1.206 [0.33]	1.229 [0.30]	−1.953 [12.71]
4	3	150	6.420 [2.20]	5.771 [2.04]	10.115 [5.08]	2.866 [0.61]	3.239 [0.65]	−13.003 [10.57]
4	3	200	10.247 [4.41]	9.316 [4.11]	9.087 [2.46]	4.733 [1.12]	5.850 [1.29]	−23.590 [13.96]
4	4	100	20.134 [9.85]	19.295 [9.65]	4.170 [2.30]	6.324 [1.93]	7.752 [2.11]	−22.588 [16.89]
4	4	150	92.005 [36.06]	89.073 [37.87]	3.188 [5.69]	25.035 [5.68]	27.216 [6.02]	−8.713 [14.92]
4	4	200	149.676 [128.74]	138.579 [129.90]	7.414 [3.94]	42.377 [15.59]	43.912 [15.47]	−3.622 [10.94]
5	2	50	0.046 [0.01]	0.099 [0.02]	−113.898 [20.57]	0.029 [0.00]	0.073 [0.01]	−154.276 [38.12]
5	2	100	0.277 [0.07]	0.358 [0.08]	−29.349 [11.60]	0.142 [0.02]	0.173 [0.01]	−22.214 [17.77]
5	2	150	0.784 [0.19]	0.861 [0.18]	−9.923 [9.57]	0.327 [0.04]	0.307 [0.03]	6.249 [16.69]
5	3	50	2.363 [0.80]	2.287 [0.79]	3.229 [5.90]	0.786 [0.20]	0.572 [0.09]	27.146 [17.02]
5	3	100	14.074 [6.50]	13.593 [6.26]	3.413 [3.15]	3.564 [1.01]	2.657 [0.59]	25.442 [11.07]
5	3	150	55.894 [25.22]	57.550 [27.14]	−2.963 [2.08]	8.430 [1.68]	7.642 [1.39]	9.348 [8.24]
5	4	50	72.060 [59.94]	77.700 [71.43]	−7.827 [14.81]	7.346 [4.79]	6.175 [3.16]	15.942 [16.57]
5	4	100	> 500	> 500	*n.a.*	35.241 [8.89]	35.016 [9.33]	0.638 [13.80]
5	4	150	> 500	> 500	*n.a.*	111.945 [34.36]	118.006 [27.95]	−5.414 [17.37]

Table 2.5: Computing time for the sequential method with intermediary filtering, the UPool method, and relative time gain as a function of $|Y^s|$ and $|S|$, for $p = 4, 5$. Average values for 20 trials.

For $p = 4$ and $p = 5$, IF benefits more from dominance filtering with CH, and we thus measure the performance improvement of UPool by considering column TG/CH. In this case, we found no positive TG, much higher variability and no monotonous trend in TG. Results using UF within UPool performed even more poorly as compared to IF using CH, and are not reported here. Thus, for these higher values of p, we cannot advise using UPool. To summarize, for $p \leqslant 3$, UPoolUF is the clear winner, whereas for $p \geqslant 4$, IFCH should be prefered.

2.4 Box-based methods

Box-based methods for speeding up the resolution of NDMSP rely on dominance relations involving combinations of *bounded boxes*. We present two dominance relations involving *combinations of boxes*. The first one is a relation between a point and a combination of boxes, and thus will be called *point-to-box* dominance (with a slight abuse since we compare the point to a *combination* of boxes, and not to a box). The second one is a relation *between* combinations of boxes, and will accordingly be called *box-to-box* dominance.

2.4.1 Box-based dominance relations

Given some set $Y \subseteq \mathbb{R}^p$, let a *box* B be defined by a subset of Y, an upper bound u such that for all y in the box, $u \geqq y$ and a lower bound l such that for all y in the box, $y \geqq l$. The tightest value for u is the ideal point of B and the tightest value for l is the anti-ideal point of B. For simplicity, we will write $y \in B$ to say that y is contained in the subset of box B, and $B \subseteq Y$ to state that all points in box B belong to Y. Let $\mathcal{B} = \{B_1, ..., B_m\}$ be a family of boxes associated with $Y \subseteq \mathbb{R}^p$. Correction of box-based filtering methods require that $\bigcup_{1 \leqslant i \leqslant m} B_i = Y$.

Consider two sets Y and Z which we want to *pool* together. *Point-to-box* dominance is defined as follows:

Definition 2.1. *Let $\bar{y} \in \mathbb{R}^p$, and two boxes $B_Y^i \subseteq Y$, $B_Z^j \subseteq Z$, with corresponding upper bounds u_Y^i and u_Z^j. $\bar{y}$* point-to-box *dominates (B_Y^j, B_Z^j) if and only if $\bar{y} \geqq u_Y^i + u_Z^j$.*

Observation 2.1. *If $\bar{y}$* point-to-box *dominates (B_Y^i, B_Z^j), then for all $y \in B_Y^i \oplus B_Z^j$, $\bar{y} \geqq y$ (see Figure 2.3). Therefore, if $\bar{y} \in Y \oplus Z$, then no point of $B_Y^i \oplus B_Z^j$ belongs to $\mathcal{N}(Y \oplus Z)$.*

We turn to the *box-to-box* relation. If combining the upper bounds of two boxes yields a point that is dominated by the combinations of lower bounds of two others boxes, then all combinations of points from the latter two boxes dominate all combinations of points from the former two boxes. Formally, we have:

Definition 2.2. *Let $B_Y^i, B_Y^j \subseteq Y$ and $B_Z^k, B_Z^l \subseteq Z$, with associated lower and upper bounds. (B_Y^i, B_Z^k)* box-to-box *dominates (B_Y^j, B_Z^l) if and only if $l_Y^i + l_Z^k \geqq u_Y^j + u_Z^l$*

Observation 2.2. *If (B_Y^i, B_Z^k)* box-to-box *dominates (B_Y^j, B_Z^l), then for all $y \in B_Y^i \oplus B_Z^k$, for all $y' \in B_Y^j \oplus B_Z^l$, $y \geqq y'$ (see Figure C.4)*

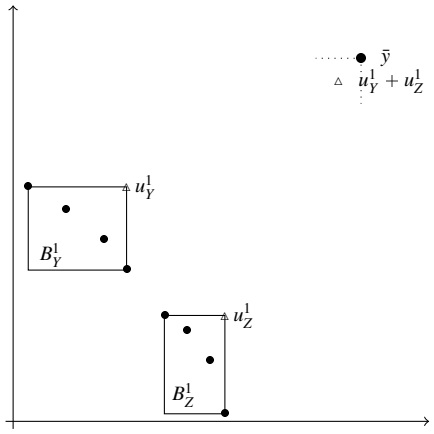


Figure 2.3: Bi-objective illustration of *point-to-box* dominance.

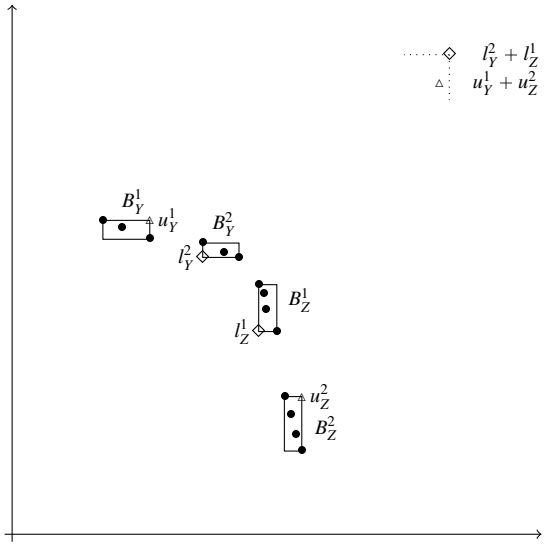


Figure 2.4: Bi-objective illustration of *box-to-box* dominance.

For efficiency, it may be desirable that the boxes be disjoint, so that dominated combinations of points are eliminated only once.

2.4.2 Algorithms for box-based dominance filtering

In a sequential approach to pooling, as described by Algorithm 11 at each step $s \in S$, we apply two boxing operations: on the one hand on $Y^{\overrightarrow{s-1}}$, the result of the previous steps, and on the other hand on Y^s, the s-th set that we want to add to the MS under construction. After removing potential combinations of points by applying either one of the box dominance relations, we apply regular dominance filtering over sums of points associated with the remaining combinations.

First, a *boxing* algorithm is called to determine how to group the contents of $Y^{\overrightarrow{s-1}}$ and of Y^s into two families of boxes, using a parameter ε as explained in the next subsection. Then we

Algorithm 11: PoolNextSetBox

input : $\overrightarrow{Y^{s-1}}, Y^s, BtB, PtB, n_\lambda, \varepsilon$
output: $\mathcal{N}(\overrightarrow{Y^{s-1}} \oplus Y^s)$

$Pooled \leftarrow \varnothing$
$(B^1_{\overrightarrow{s-1}}, ..., B^n_{\overrightarrow{s-1}}) \leftarrow$ **Boxing**$(\overrightarrow{Y^{s-1}}, \varepsilon)$
$(B^1_s, ..., B^m_s) \leftarrow$ **Boxing**(Y^s, ε)
$BoxCombinations \leftarrow \{B^1_{\overrightarrow{s-1}}, .., B^n_{\overrightarrow{s-1}}\} \times \{B^1_s, ..., B^m_s\}$
if PtB **then**
  PointFiltBox$(BoxCombinations, n_\lambda)$
if BtB **then**
  BoxFiltBox$(BoxCombinations)$
for $(B^i_{\overrightarrow{s-1}}, B^j_s) \in BoxCombinations$ **do**
  for $y \in B^i_{\overrightarrow{s-1}}$ **do**
    for $z \in B^j_s$ **do**
      $Pooled \leftarrow Pooled \cup \{y + z\}$

apply either *point-to-box* or *box-to-box* filtering, depending on the values of booleans PtB and BtB respectively.

2.4.2.1 Point-to-box dominance

To filter according to the *point-to-box* dominance relation, we need a pool $P \subset \overrightarrow{Y^{s-1}} \oplus Y^s$ of point combinations. As described in Algorithm 12, we propose to use supported points, by optimizing positively weighted sums for each sets, using weight set $\Lambda \subseteq \mathbb{R}^p_{>0}$ generated according to parameter n_λ, to produce supported points. Points of the resulting pool will be compared to the combinations of upper bounds of boxes in Algorithm 13.

The points whose values maximize some positive weight sum are of course non-dominated in the sets in which they are found. Combining two points which are non-dominated in their respective sets Y and Z can yield a dominated point in the set sum $Y \oplus Z$. In this case, however, we use the fact that for any sets Y, Z and $\lambda \in \Lambda$,

$$\max\{\lambda(y+z) \mid y+z \in Y \oplus Z\} = \max\{\lambda y \mid y \in Y\} + \max\{\lambda z \mid z \in Z\}$$

to guarantee that these points are not only feasible but also non-dominated, i.e. they belong to $\mathcal{N}(\overrightarrow{Y^{s-1}} \oplus Y^s)$ and can be immediately included in the set sum under construction.

A key step in producing supported points is generating the set of weights defining the scalarizations for which supported points are optimal. A good set of weights would produce points which evenly cover $\overrightarrow{Y^{s-1}} \oplus Y^s$, and do not lead to too many redundant points. Because generating those supported points takes place within a box-based dominance filtering algorithm which must

be quick, we must also bound the number of weight vectors we want to consider.

In the bi-objective case, we can easily generate the set of supported points exhaustively using the dichotomic method by Aneja and Nair (1979), and an ε-dominance based threshold to stop the exploration of a weight-space interval and thus control for the desired number of weights vectors. Beyond the bi-objective case, there is no systematic and scalable method enumerating the whole set of supported points. Following Borges and Hansen (2002), we could generate sets of maximally dispersed weights, so as to hopefully reduce the number of redundant points. However, experimentally, we found that this approach did not yield better performance than generating random weight vectors.

The number of weight vectors to be generated is denoted by n_λ, which we tuned manually. We produce supported points according to the computed set of weights, and we remove duplicates, i.e. points that were supported by several different weight vectors. Then for each box combination, we iterate over these supported points and test for *point-to-box* dominance between the supported point and the box combination, as described by Algorithm 13.

Algorithm 12: getSupported

input : $Y^{\overrightarrow{s-1}}, Y^s, \Lambda$
output: $Supported$

$Supported \leftarrow \varnothing$
for $\lambda \in \Lambda$ **do**
  $y^1 \leftarrow \max\{\langle \lambda, y \rangle \mid y \in Y^{\overrightarrow{s-1}}\}$
  $y^2 \leftarrow \max\{\langle \lambda, y \rangle \mid y \in Y^s\}$
  $y^* \leftarrow y^1 + y^2$
  if $y^* \notin Supported$ **then**
    $Supported \leftarrow Supported \cup \{y^*\}$
return $Supported$

Algorithm 13: PointFiltBox

input : $BoxCombinations, n_\lambda$
output: $BoxCombinations$

$\Lambda \leftarrow$ **generateWeightSet**(n_λ)
$P \leftarrow getSupported(Y^{\overrightarrow{s-1}}, Y^s, \Lambda)$
for $(i, j) \in BoxCombinations$ **do**
  for $\bar{y} \in P$ **do**
    if $\bar{y} \succeq u^{Y_i^{\overrightarrow{s-1}}} + u^{Y_j^s}$ **then**
      $BoxCombinations \leftarrow BoxCombinations \backslash (i, j)$
return $BoxCombinations$

2.4.2.2 Box-to-box dominance

For filtering according to the *box-to-box* dominance relation, we adapt the unidirectional dominance filtering algorithm (UF) to the case of comparing box combinations. Correction of a UF requires the input to be ordered in a *dominance preserving order*. It is easily seen that the decreasing lexicographic order over sums of upper bounds of combinations of boxes is a dominance preserving order for *box-to-box* dominance:

Observation 2.3. *Let* $B_Y^1, B_Y^2 \subseteq Y$, *and* $B_Z^1, B_Z^2 \subseteq Z$. *If* $u_Y^1 + u_Z^1 \succeq_{Lex} u_Y^2 + u_Z^2$, *then* $l_Y^2 + l_Z^2 \not\succeq u_Y^1 + u_Z^1$

Algorithm 13 describes the pseudo-code for the *box-to-box* filtering procedure. $\succeq_{Lex}^{u}$ denotes the decreasing lexicographic order over the sums of upper bounds of combinations of boxes, i.e. for (B_Y^1, B_Z^1), and (B_Y^2, B_Z^2) two pairs of boxes, by definition, $(B_Y^1, B_Z^1) \succeq_{Lex}^{u} (B_Y^2, B_Z^2)$ if and only if $u_Y^1 + u_Z^1 \succeq_{Lex} u_Y^2 + u_Z^2$.

If $p = 2$, we use UF with a unique dominance comparison with the last combination of boxes found to be non-dominated. For $p \geqslant 3$, we adapt the approach by Chen et al. (2012) of insertion into a KDTree to check whether an input box combination is dominated or dominates previously considered box combinations. We will denote generically by $CheckDtdIn(y, A)$ an algorithm which determines whether some box combination y is dominated by an element of set A.

Once *box-to-box* filtering has been performed, we are left with a set of non-dominated pairs representing combinations of boxes. We iterate over these pairs, and for each pair of boxes, we compute the MS of their contents, and append it to *Pooled*. Finally we apply a dominance filtering algorithm to *Pooled* before returning it.

2.4.3 An algorithm for creating a family of boxes

A variety of algorithms can be used to produce a family of boxes from an input set of points. As preliminary work, we have compared five approaches to achieve that end. The first one consists in a quantization of each set: for each dimension, a desired *coarseness* is defined, and the orthogonal projection of the set onto this dimension is divided into intervals of equal length. Each point then belongs to a unique box which is defined by an interval on each dimension. The second one consists in defining a desired number of boxes, and using either ascendant hierarchical or k-means clustering to define k boxes. The third one consists in defining a minimum and a maximum number of points per box, and inserting points into their closest box until the closest box is full. When a box is full, we split it according to a rule that ensures that the new boxes contain the minimum number of points, and makes the two new boxes as distinct as possible. The fourth approach consists in defining a threshold distance, considering each point in the input set in turn, and adding it to the closest already defined box which lies at a distance smaller than the threshold. If such a box does not exist, a new box is created for the current point and the process continues. A fifth approach would be similar to the fourth one, using not a distance threshold but ε-dominance to

Algorithm 14: BoxFiltBox

input : *BoxCombinations*
output: *NDBoxComb*

/*Assume that $(i, j) \in BoxCombinations$ is the combination of B_Y a box of elements of Y, and B_Z a box of elements of elements of Z, where $Y \leftarrow \overrightarrow{Y^{s-1}}$ and $Z \leftarrow Y^s$. */
$NDBoxComb \leftarrow \varnothing$
$sort(BoxCombinations, \geq_{Lex})$
for $(i, j) \in BoxCombinations$ **do**
  $Dtd \leftarrow False$
  if $p = 2$ **then**
    /*Assume that $NDBoxComb = \{(B_Y^1, B_Z^1), ..., (B_Y^m, B_Z^n)\}$ */
    if $l_Y^m + l_Z^n \geq u_Y^i + u_Z^j$ **then**
      $Dtd \leftarrow True$
  else
    $Dtd \leftarrow CheckDtdIn((B_Y^i, B_Z^j), NDBoxComb)$
  if $not(Dtd)$ **then**
    $NDBoxComb \leftarrow NDBoxComb \cup (B_Y^i, B_Z^j)$
return $NDBoxComb$

determine whether two points should belong to the same box.

We will only describe in details, and provide experimental results for the fifth approach to creating boxes, based on ε-dominance, and we begin with defining the notion.

Definition 2.3. *For all $y, y' \in Y$, y is said to ε-dominate y', written $y \geq_\varepsilon y'$ if and only if*

$$y_j(1 + \varepsilon) \geqslant y'_j \quad \forall j \in \{1, ..., p\}$$

Moreover, y is said to be ε-indifferent to y', written $y \sim_\varepsilon y'$ if and only if

$$y \geq_\varepsilon y' \text{ and } y' \geq_\varepsilon y$$

The *boxing* algorithm operates as follows. Consider a first point $y \in Y$, and create a box containing y. Then for each remaining point $y' \in Y$, check whether $y \sim_\varepsilon y'$. If it is the case, add y' to the box associated with y, and remove y' from Y. If not, do nothing. When all of $Y \backslash \{y\}$ has been examined, remove y from Y and repeat the process with a new head of list. This procedure is formally described as Algorithm 15.

2.4.4 Experimental Results

In the next experiment, we measure the time gain (TG) yielded by the use of box-based methods in solving NDMSP, and the average number of points eliminated by *each box − based* test (*av.*

Algorithm 15: Boxing using ε-dominance

input : $Y \subseteq \mathbb{R}^p, \varepsilon \in \mathbb{R}$
output: $(B_1, ..., B_m)$

```
Boxes ← ∅
while Y ≠ ∅ do
    Choose y ∈ Y
    Let B_y be a new box containing y.
    for y' ∈ Y\{y} do
        if y ∼_ε y' then
            B_y ← B_y ∪ {y'}
        Y ← Y\{y'}
    Boxes ← Boxes ∪ {B_y}
    Y ← Y\{y}
return Boxes
```

Av). The goal of the latter measurement is to contrast the ways in which *box-to-box* and *point-to-box* dominance filtering yield time gain. We consider *box-to-box* and *point-to-box* independently. Instances are identical to those used in the previous experiment.

We tuned ε so as to get the best possible results. Preliminary experiments suggest that decreasing the value of ε as $|Y|$ increases allows better results. However, as we were not able to analyze this dependency compellingly, we resorted to the use a single value of ε for each value of p. In this case, for *box-to-box* dominance we used $\varepsilon = 0.001$ for $p = 2$, $\varepsilon = 0.02$ for $p = 3$, $\varepsilon = 0.06$ for $p = 4$ and $\varepsilon = 0.1$ for $p = 5$. For *point-to-box* dominance, we used $\varepsilon = 0.015$ for $p = 2$ and $\varepsilon = 0.03$ for $p = 3$. This dominance relation requires us to choose a number of weight vectors to compute. Intuitively, it seems that n_λ should vary with the number of "feasible points" at one step of the computation, i.e. with $|\overrightarrow{Y^{s-1}} \times Y^s|$. However we were not able to find a relationship between $|\overrightarrow{Y^{s-1}} \times Y^s|$ and n_λ for which performance was significantly better than when using a fixed value of n_λ.

Results for *box-to-box* dominance, in Tables 2.6 and 2.7, suggest that TG does increase as $|Y^s|$ increases. For $p = 2$, we found maximum TGs ranging from about 40.51 to about 60.74%, suggesting that *box-to-box* dominance is effective at speeding up the resolution of NDMSP. However, an increase in $|S|$ appears to "flatten" the TG profile, and limit performance especially for higher values of $|Y^s|$. This is most likely due to the use of a single parameter for boxing, which becomes unfit to larger set sizes. For $p = 3$, we find more moderate TG. We observe, however, a more pronounced increase of performance as $|Y^s|$ increases, which suggests that scaling is probably still good, and thus the approach viable, for this number of criteria. For $p = 4$ however, we find small TG, high variability, and no clear trend as S or $|Y^s|$ increases, thus cannot claim that the method is viable in this case. For $p = 5$, we found no positive time gain.

Note that for $p = 2$ and $p = 3$ (Table 2.6), box-based filtering with *box-to-box* dominance performed worse than the best version of UPool in this case (column $TG/UPUF$), by a large margin. However, for $p = 4$ (Table 2.7), where UPool tended to perform rather worse than IF,

p	$\|S\|$	$\|Y^s\|$	$IF\ UF$		$Box-to-Box$							
			$Time$		$Time$		TG/UF		$av.Av$		$TG/UPUF\ (\%)$	
2	2	200	0.197	[0.05]	0.117	[0.03]	40.513	[10.79]	1.686	[0.10]	−62.188	[23.45]
2	2	400	0.466	[0.12]	0.184	[0.06]	60.586	[14.60]	3.040	[0.17]	−39.152	[44.54]
2	2	600	0.923	[0.06]	0.362	[0.06]	60.737	[7.26]	4.572	[0.36]	−57.590	[27.38]
2	3	200	0.314	[0.06]	0.166	[0.04]	47.056	[8.93]	2.348	[0.18]	−68.798	[25.70]
2	3	400	1.294	[0.30]	0.620	[0.34]	52.112	[14.42]	4.322	[0.45]	−79.141	[51.54]
2	3	600	4.357	[1.59]	1.802	[0.56]	58.649	[11.43]	6.770	[0.80]	−109.336	[46.55]
2	4	200	0.952	[0.38]	0.562	[0.22]	40.980	[13.07]	3.183	[0.39]	−133.188	[41.00]
2	4	400	5.178	[2.16]	2.850	[1.94]	44.959	[23.14]	5.972	[1.00]	−169.858	[95.96]
2	4	600	11.534	[3.19]	6.700	[3.02]	41.914	[12.05]	8.716	[1.07]	−216.514	[63.98]
3	2	200	0.158	[0.03]	0.163	[0.03]	−3.141	[9.95]	2.244	[0.15]	−41.041	[11.24]
3	2	300	0.339	[0.04]	0.305	[0.06]	9.873	[10.32]	3.054	[0.22]	−38.606	[12.43]
3	2	400	0.668	[0.09]	0.498	[0.13]	25.496	[11.08]	3.874	[0.28]	−26.574	[12.87]
3	3	200	1.366	[0.51]	1.339	[0.51]	1.922	[12.48]	5.123	[0.69]	−47.850	[23.87]
3	3	300	2.902	[0.85]	2.196	[0.77]	24.325	[9.91]	7.363	[1.06]	−42.022	[13.67]
3	3	400	5.886	[1.77]	4.046	[1.82]	31.261	[11.71]	9.381	[1.03]	−34.296	[13.83]
3	4	200	7.049	[3.11]	6.870	[3.42]	2.534	[9.91]	9.464	[2.02]	−51.938	[16.54]
3	4	300	15.396	[4.78]	12.854	[5.87]	16.509	[14.76]	13.467	[2.46]	−54.079	[17.40]
3	4	400	29.178	[12.22]	23.715	[14.31]	18.723	[14.29]	15.829	[3.62]	−60.398	[15.73]

Table 2.6: Computing time for the sequential method with intermediary filtering, for *box-to-box* dominance, and relative time gain as a function of $|Y^s|$, for $p = 2, 3$. Average values for 20 trials.

p	$\|S\|$	$\|Y^s\|$	IFCH		$Box-to-Box$							
			$Time$		$Time$		TG/CH		$av.Av$		$TG/UPCH\ (\%)$	
4	2	100	0.082	[0.02]	0.081	[0.01]	0.081	[11.74]	2.346	[0.13]	16.477	[11.25]
4	2	150	0.184	[0.03]	0.174	[0.03]	0.174	[14.12]	2.803	[0.25]	19.223	[13.30]
4	2	200	0.320	[0.05]	0.293	[0.09]	0.293	[19.14]	3.404	[0.43]	20.426	[16.97]
4	3	100	1.206	[0.33]	1.214	[0.37]	1.214	[17.04]	3.490	[0.67]	1.273	[19.42]
4	3	150	2.866	[0.61]	2.746	[0.62]	2.746	[11.10]	3.881	[0.97]	15.222	[8.97]
4	3	200	4.733	[1.12]	4.667	[1.17]	4.667	[12.20]	3.846	[1.27]	20.221	[10.24]
4	4	100	6.324	[1.93]	6.384	[1.80]	6.384	[10.63]	2.328	[0.46]	17.646	[11.06]
4	4	150	25.035	[5.68]	24.228	[5.26]	24.228	[11.74]	2.458	[0.65]	10.981	[11.05]
4	4	200	42.377	[15.59]	41.476	[16.14]	41.476	[6.96]	2.979	[1.18]	5.548	[11.44]

Table 2.7: Computing time for the sequential method with intermediary filtering, for *box-to-box* dominance, and relative time gain as a function of $|Y^s|$, for $p = 4$. Average values for 20 trials.

$\|S\|$	$\|Y^s\|$	*IF UA*		*Point − to − Box*									
		Time		*Time*		*TG/UF*		*av. Av*		n_λ		*TG/UPUF (%)*	
2	200	0.197	[0.05]	0.100	[0.05]	49.098	[19.05]	30.667	[6.82]	9.150	[2.61]	−38.781	[51.13]
2	400	0.466	[0.12]	0.207	[0.07]	55.545	[18.76]	102.907	[32.94]	10.350	[2.51]	−56.948	[45.88]
2	600	0.923	[0.06]	0.401	[0.14]	56.581	[15.94]	159.007	[48.80]	10.550	[2.72]	−74.271	[63.22]
3	200	0.314	[0.06]	0.140	[0.05]	55.379	[11.93]	64.344	[12.70]	13.350	[2.59]	−42.263	[33.19]
3	400	1.294	[0.30]	0.552	[0.21]	57.367	[12.26]	172.334	[53.46]	14.400	[3.08]	−59.483	[42.72]
3	600	4.357	[1.59]	1.420	[0.48]	67.419	[14.60]	381.415	[103.48]	14.900	[2.35]	−64.939	[48.52]
4	200	0.952	[0.38]	0.443	[0.14]	53.467	[13.17]	109.464	[28.79]	15.800	[3.57]	−83.852	[33.32]
4	400	5.178	[2.16]	1.987	[1.08]	61.633	[12.08]	344.285	[118.76]	17.700	[2.93]	−88.108	[44.97]
4	600	11.534	[3.19]	5.148	[2.15]	55.368	[9.49]	655.786	[188.35]	18.150	[3.73]	−143.201	[43.58]

Table 2.8: Computing time for the sequential method with intermediary filtering, for *point-to-box* dominance, and relative time gain as a function of $|Y^s|$, for $p = 2$. $Sup\ Ef = 100$ in each case. Average values for 20 trials.

$\|S\|$	$\|Y^s\|$	*IF UA*		*Point − to − Box*									
		Time		*Time*		*TG/UF*		*av. Av*		*Sup Ef*		*TG/UPUF (%)*	
2	200	0.158	[0.03]	0.168	[0.03]	−6.345	[12.77]	3.172	[0.61]	21.050	[5.06]	−45.423	[15.16]
2	300	0.339	[0.04]	0.310	[0.07]	8.604	[12.96]	5.310	[1.20]	20.450	[4.03]	−40.558	[19.40]
2	400	0.668	[0.09]	0.600	[0.13]	10.146	[11.92]	6.463	[1.46]	20.350	[4.03]	−52.652	[19.84]
3	200	1.366	[0.51]	1.313	[0.52]	3.892	[4.95]	8.697	[2.08]	29.400	[6.03]	−44.881	[9.77]
3	300	2.902	[0.85]	2.592	[0.86]	10.675	[9.14]	11.960	[3.49]	29.200	[5.22]	−67.640	[16.74]
3	400	5.886	[1.77]	5.003	[1.92]	15.003	[11.37]	18.678	[4.96]	28.650	[5.07]	−66.059	[19.23]
4	200	7.049	[3.11]	6.840	[3.18]	2.956	[5.01]	14.164	[3.93]	41.050	[5.47]	−51.281	[10.41]
4	300	15.396	[4.78]	14.240	[4.96]	7.504	[5.72]	21.873	[9.11]	40.100	[4.78]	−70.698	[11.06]
4	400	29.178	[12.22]	26.724	[12.64]	8.409	[6.78]	28.136	[9.50]	37.850	[4.70]	−80.751	[18.89]

Table 2.9: Computing time for the sequential method with intermediary filtering, for *point-to-box* dominance, and relative time gain as a function of $|Y^s|$, for $p = 3$. $n_\lambda = 100$ in all instances. Average values for 20 trials.

we found that it is also supplanted by *box-to-box* (column $TG/UPCH$). However, *box-to-box* itself performs very similarly to IF or worse, so that IF should still be considered the most reliable approach for $p = 4$.

Turning to results for *point-to-box* dominance, in Tables 2.8 and 2.9, when $p = 2$, we found the best results when successful box-based-dominance tests tend to eliminate many points, suggesting that the optimal box size is larger. We found *point-to-box* dominance filtering to yield significant TG, from about 49.1 up to about 67.42%. Performance did not appear to vary significantly with increases in $|Y^s|$ or in $|S|$. It remains to be investigated whether this is due to tuning issues or a need for adapting the parameter values with increasing $|Y^s|$ and $|S|$. For $p = 3$, we found slightly positive TG, although variability remained too high to warrant that this TG will always be positive. For $p = 4$, TG was always negative is not reported.

Although one may expect the optimal number of boxes to be closely dependent on the number of supported points generated, we found that for $p = 2$, it was nearly always beneficial to generate the complete set of supported points using the dichotomic method. For $p \geqslant 3$, we were not able to find a very stable relationship between n_λ and TG. We tuned n_λ by hand and measured the ratio of unique supported points to the number of generated weight vectors, and we found that about 20% of successful generation (column *Sup. Ef*) of supported points was associated with highest TG. Smaller n_λ will yield higher *Sup. Ef* but smaller TG, probably by lack of diversity or coverage by

supported points necessary for *point-to-box* filtering. Higher n_λ will put additional burden on the filtering operation. Whether we should expect optimal n_λ to increase with $|\overrightarrow{Y^{s-1}} \times Y^s|$ is in our opinion not obvious - as the proportion of supported points depends on other factors than the sheer number of points, and we could not confidently confirm it.

In all reported cases, box-based dominance filtering using *point-to-box* dominance performed worse than UPool (column $TG/UPUF$). Thus as a conclusion to this subsection, we note that both *box-to-box* and *point-to-box* dominance filtering appear to be promising tools for speeding up the resolution of NDMSP when $p = 2$, and that *box-to-box* dominance filtering remains reliable when $p = 3$, but beyond, it is beaten by the simple IF approach.

UPool lost only against *box-to-box* for $p = 4$, but in this context, both methods performed slightly worse than IF. In all other cases, UPool performed similarly or better than *box-based* dominance approaches, and should thus be preferred.

2.5 Conclusions and discussion

When computing the non-dominated subset of the Minkowski Sum of a family of set, a great proportion of the sums of points generated at intermediary steps of computing the MS end up being dominated in the final result. This is why, as we have seen, filtering after having added each individual set reduces computing time hugely. We have sought to further decrease the resolution time of NDMSP by reducing the number of comparisons made at each intermediary step of the pooling process.

To this end, we proposed a unidirectional pooling algorithm, UPool, based on the lexicographic sorting of the input sets. This approach seeks to improve computation time relative to unidirectional dominance filtering algorithms by significantly saving on the initial sorting of the set sum. We also proposed using bound reasoning over boxes, i.e. subsets of the sets to be summed together. We defined two dominance relations: *point-to-box* and *box-to-box*. For $p = 2$, filtering methods based on these two dominance relations have shown promising results, and the methods appeared to scale with the size of the sets. For $p = 3$ however, only *box-to-box* dominance showed reliable benefits, and beyond, no box-based approach performed better than UPool or IF. Although we investigated the combining of both dominance relations, because optimal parameter values for creating boxes seem to differ between the two, we could not, so far, draw conclusive results, and further research on this topic is needed.

The main advantage of UPool as opposed to *box-based* methods is that it requires no parameter tuning on which the scaling of performance improvement with regards to size or number of criteria would depend. Moreover, in all presented experiments, UPool appeared to be the preferable option, as it performed better than *box-based* approaches in all cases but for $p = 4$, the basic approach IF performed slightly better. In the next chapters, we will use UPool (adapted to the combination of solutions, not just points) as our default pooling method.

An immediate development of our work on box-based dominance relations should provide a *boxing* algorithm with parameters that are independent of the value of p. One line of research would consider a trade-off between two criteria. The first one would be a desired minimum number of points per box, so as to control the number of points eliminated per box-based dominance test. The second one would be the maximum acceptable distance, over criteria, between a point and the box it belongs to, so as to keep the upper bound of a box tight enough.

Chapter 3

Decoupling a coupled problem to obtain bound sets [1]

Chapter Abstract

In this chapter, we explore the notion of decomposition in multiobjective optimization. Because we deal with complex system problems which are not straightforwardly decomposable, we have to modify their original formulation to decouple them. This results in *restricted* problems, which provide *lower* bound sets to the non-dominated set, and *relaxations*, which provide *upper* bound sets to the non-dominated set. Decomposition allows the resolution of these decomposable variants in a fraction of the time needed to solve the original problem, an advantage that increases significantly with the number of dimensions of the objective space.

Contents

3.1 Introduction 60
3.1.1 Bound sets 60
3.1.2 Multiobjective relaxations and restrictions 61
3.2 Decomposable restrictions and lower bound sets 63
3.2.1 Admissible variable values, neutral variable values and restrictions . . 63
3.2.2 Restrict-splitting coupling constraints 64
3.3 Decomposable relaxations and upper bound sets 65
3.3.1 Copy-splitting coupling constraints 66
3.3.2 Local relaxation of coupled variables in subproblem constraints 66
3.4 Experimental results 68
3.5 Conclusions and discussion 74

[1] This chapter is based on submitted article Kerbérénès et al. (2021a)

3.1 Introduction

In this chapter, we show how decomposability can be exploited in solving coupled MOCO problems. This is not straightforward since, typically, a coupled system is not such that $\prod_{s\in S} X^s = X$, where S is the set of subsystems and X^s the feasible set of the subproblem associated with subsystem $s \in S$. While the original formulation may not be uncoupled, we can obtain decoupled variants of it by generic modifications. This chapter describes a number of these variants. Each of them is based either on a relaxation $X' \supseteq X$ or a restriction $X'' \subseteq X$ of the feasible set X. The following enumerates these variants:

- Ignoring coupling constraints yields a relaxation of the original problem.
- Restricting the original problem to non-coupled variables yields a restriction of the original problem.
- *Restrict-splitting* coupling constraints, by which we mean fixing, for each coupling, the variables associated with each subsystem except one, yields a restriction of the original problem.
- *Copy-splitting* coupling constraints, by which we mean replacing each coupling constraint with $|S|$ restrictions of itself, one to each subsystem $s \in S$, yields a relaxation of the original problem.
- More surprisingly, ignoring coupled variables in subsystems local constraints also yields an uncoupled problem. In this variant, the subsystems problems become uncoupled, and the coupled variables, bound only by coupling constraints, form a new, independent subproblem. The result is a relaxation of the original problem.

3.1.1 Bound sets

While the original formulation of an MOP may not be uncoupled, we can obtain decoupled variants of it through generic modifications. The set of non-dominated points associated with each of these variants will *bound* the set of images of efficient solutions of the original problem from either above or below.

3.1.1.1 Definitions

An application of relaxations is to obtain *upper bound sets*, and an application of restrictions is to obtain *lower bound sets*, which we will now define. The image y of an efficient solution of a relaxation delineates a zone of the objective space which is "unreachable", in the sense that no feasible solution can be found to dominate y. Assuming we are maximizing we give the following definitions, for $\mathbb{R}^p_{\geq} = \{y \in \mathbb{R}^p \mid y \geq \mathbf{0}\}$:

Definition 3.1. *Point $y \in \mathbb{R}^p$ is an upper bound point on $Y \subseteq \mathbb{R}^p$ if and only if*

$$\nexists y' \in Y, y' \succeq y$$

or equivalently if $Y \cap y \oplus \mathbb{R}^p_{\geq} = \varnothing$. A set of upper bound points will be called a weak upper bound set.

The union of the unreachable zones defined by the elements of a weak upper bound set defines an unreachable region, which is especially useful in the context of Branch & Bound dominance relations.

Definition 3.2. *Point $y \in \mathbb{R}^p$ is a lower bound point on $Y \subseteq \mathbb{R}^p$ if an only if*

$$\nexists y' \in Y, y \succeq y'$$

or equivalently if $Y \cap y \ominus \mathbb{R}^p_{\geq} = \varnothing$. A set of lower bound points will be called a weak lower bound set.

Definition 3.3. *(Ehrgott and Gandibleux (2001)) Given $Y \subseteq \mathbb{R}^p$, UB is a strong upper bound set to Y (in the maximization case) if and only if*

$$\begin{cases} \forall y \in Y \ \exists y' \in UB, y' \succeq y \\ \forall y \in UB, y \text{ is an upper bound point to } Y \end{cases} \tag{3.1}$$

or equivalently if $Y \subset UB \ominus \mathbb{R}^p_{\geq}$

In other words, in addition to ensuring that a strong upper bound set delineates an unreachable region, we also require that every point of the set which we want to bound from above be dominated by or equal to some element of the strong upper bound set.

Definition 3.4. *(Ehrgott and Gandibleux (2001)) For any $Y \subseteq \mathbb{R}^p$, LB is a strong lower bound set to Y (in the maximization case) if and only if*

$$\begin{cases} \forall y \in Y \ \exists y' \in LB, y \succeq y' \\ \forall y \in LB, y \text{ is a lower bound point to } Y \end{cases} \tag{3.2}$$

or equivalently if $Y \subset LB \oplus \mathbb{R}^p_{\geq}$.

Example 3.1. *Figure C.5 provides an example of a set $N = \{n^i \mid i = 1, ..., 5\}$ which is weakly bounded from below by set $L = \{l^i \mid i = 1, ..., 5\}$, because n^3 dominates no element in L. However $U = \{u^i \mid i = 1, ..., 5\}$ is a strong upper bound on N.*

3.1.2 Multiobjective relaxations and restrictions

Upper bound sets can be obtained by solving relaxations of the original problem, and lower bound sets by solving restrictions of the same problem. Informally, a relaxation is defined by replacing a collection of constraints of the original problem with a less constraining collection of constraints.

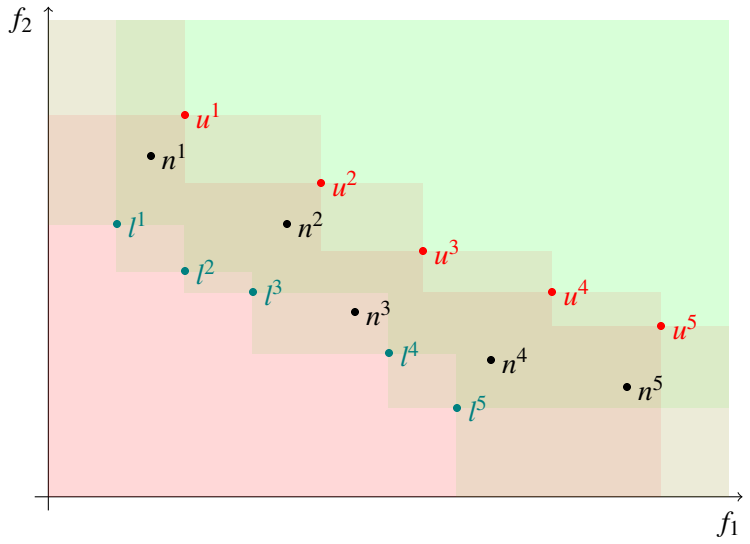


Figure 3.1: Let $N = \{n^i \mid i = 1, ..., 5\}$, $L = \{l^i \mid i = 1, ..., 5\}$, $U = \{u^i \mid i = 1, ..., 5\}$. L is a *weak* lower bound on N, U is a strong upper bound on N

Definition 3.5. *(Nemhauser and Wolsey, 1998) For f and f' taking values in $\mathbb{R}$, $\max_{x' \in X'} f'(x')$ is a relaxation of $\max_{x \in X} f(x)$ if and only if $X \subseteq X'$ and for any $x \in X$, $f(x) \leqslant f'(x)$.*

We propose the following extension of the notion to the multiobjective case:

Definition 3.6. *$\max_{x' \in X'}(f'_1(x'), ..., f'_p(x'))$ is a multiobjective relaxation of $\max_{x \in X}(f_1(x), ..., f_p(x))$ if and only if for all $j \in \{1, ..., p\}$, $\max_{x' \in X'} f'_j(x')$ is a relaxation of $\max_{x \in X} f_j(x)$*

A multiobjective relaxation yields a feasible set that is a superset of the original feasible set, so that no efficient solution of the relaxation can be dominated by a solution in the original problem.

Proposition 3.1. *Let $f(x) = (f_1(x), ..., f_p(x))$ where f_j takes values in $\mathbb{R}$ for each $j \in \{1, ..., p\}$. If $\max_{x' \in X'} f'(x')$ is a relaxation of $\max_{x \in X} f(x)$, then $\mathcal{N}(f'(X'))$ is an upper bound set on $\mathcal{N}(f(X))$.*

Proof. Assume by contradiction that there exists $y' = f'(x') \in \mathcal{N}(f'(X'))$ and $y = f(x) \in \mathcal{N}(f(X))$, such that $y \geq y'$. Then because $X \subseteq X'$, we have $x \in X'$, and thus $f'(x) \geqq f(x) \geq f'(x')$, so that $y' \notin \mathcal{N}(f'(X'))$, contradicting our assumption. □

A restriction of a problem with feasible set X is simply a variant with feasible set $X' \subseteq X$.

Observation 3.1. *For any $X' \subseteq X$, $\mathcal{N}(f(X'))$ is a lower bound set to $\mathcal{N}(f(X))$.*

Proof. Assume that there is $y' = f(x') \in \mathcal{N}(f(X'))$, and $y = f(x) \in \mathcal{N}(f(X))$ such that $y' \geq y$. Because $X' \subseteq X$, we have $x' \in X$, contradicting the assumption that $y \in \mathcal{N}(f(X))$. □

In the next two sections, we formally describe methods for obtaining variants of the original problem, each of which being either a relaxation, with a feasible set $X' \supseteq X$, or a restriction, with a feasible set $X'' \subseteq X$, of the original feasible set X, while leading to a decomposable problem.

3.2 Decomposable restrictions and lower bound sets

3.2.1 Admissible variable values, neutral variable values and restrictions

In order to obtain an uncoupled restriction of the original problem, we need to fix all, or, as we shall see, *most* coupling variables to some *admissible* or *neutral* values. Given a feasible solution $x \in X$, we denote by x_{-i} the vector decision variables where variable x_i is omitted. Therefore, up to some permutation, we have $x \sim (x_{-i}, x_i)$. When focusing on a subset of $I \subseteq N$ of variables, we write $x \sim (x_{\bar{I}}, x_I)$.

Definition 3.7. *Value e_i is an admissible value for variable x_i if there exists $x \in X$ such that $x_i = e_i$, or equivalently $(x_{-i}, e_i) \in X$.*

Put simply, a value is *admissible* for a decision variable if we know that there exists a feasible solution in which the decision variable takes this value. This will help us fix the coupling to one possible interaction between the subsystems, considering one of the possible ways in which this interaction could occur, while maintaining the feasibility of solutions obtained under this assumption. A stronger notion is that of a *neutral* value assignment, by which we denote a value assignment that is feasible for any other variables assignment:

Definition 3.8. *Value e_i is a neutral value for variable x_i if, for all $x \in X$, $(x_{-i}, e_i) \in X$.*

Observation 3.2. *Let e_i be a neutral value for variable x_i, then $e_N := (e_i \mid i \in N) \in X$.*

For example, in the knapsack problem (KP), it is easily seen that some variable assignments do not alter the feasibility of a solution, and are thus *neutral*. Consider a partial feasible solution to KP, and an item i for which no decision has been made. Not taking the item, i.e. setting the corresponding decision variable x_i to 0, can never yield an infeasible solution. But in general, we cannot assume that assigning the value 0 to a variable preserves feasibility.

Obviously, as long as the feasible set is non-empty, any neutral value is also an admissible value. Admissible values always exist, but must be determined as the corresponding variable value of a feasible solution, which implies that we find such a feasible solution. On the contrary, neutral values may not always exist, but can be substituted into the original problem formulation without having to first find a feasible solution. This is an advantage when a neutral value is known for a whole class of problem, such as 0 for any variable in an instance of KP.

3.2.2 Restrict-splitting coupling constraints

Consider again the formulation of the original problem, with feasible set X:

$$\begin{array}{lll} \max & f(x) & \\ s.t. & x^k \in X^k & \forall\, k \in K \\ & x^s \in X^s & \forall\, s \in S \end{array} \qquad (P)$$

Let $x^s_K = (x^s_k \mid k \in K)$ denote the vector of variables in subsystem s which are coupled by some coupling constraint. Thus $x^s \sim (x^s_K, x^s_{\bar{K}})$, with $x^s_{\bar{K}}$ the vector of uncoupled variables in s. Then we will denote by $x_{\bar{K}}$ the vector of all uncoupled variables.

The simplest way to obtain a decomposable restriction to the original problem is to fix all variables which appear in coupling constraints to an admissible value, or possibly a neutral value. Then, because no free variable remains linked by some constraint to free variables from other subsystems, the problem is obviously decomposable. However, the larger the proportion of coupled variables in the original problem, the poorer the quality of approximation of the non-dominated set of this original problem provided by the non-dominated set of this modified problem.

Intuitively, by fixing all coupled variables, we prevent any interaction between subsystems, we will call this weak lower bound the *local restriction* or *LocRes* lower bound. We can however, obtain decomposition by fixing only a subset of the set of coupled variables. Thus, in *restrict-split* variants of the original problem, we choose, for each $k \in K$, one subsystem s coupled by constraint k. We fix all coupled variables appearing in coupling constraint k, except those which belong to subsystem s. In this case, even if there were no strictly local variables, the restriction would not be reduced to a single feasible solution.

Let $K_s \subseteq K$ be the subset of coupling constraints which involve variables from subsystem s, and in which we have chosen to keep free variables x_s, while all other variables are set to some admissible value. For $k \in K_s$, we denote by x^s_k the variables of subsystem s which appear in constraint k, and by $x^{\bar{s}}_k$ the variables of all subsystems except s, which appear in constraint k. Let x_{K_f} be the subvector of coupled variables which are left free. The *restrict-split* variant of problem (P) is:

$$\begin{array}{lll} \max & f(x_{\bar{K}}, x_{K_f}, e_{\overline{K_f}}) & \\ s.t. & (x^s_k, e^{\bar{s}}_k) \in X^k & \forall s \in S, \forall\, k \in K_s \\ & x^s \in X^s & \forall\, s \in S \end{array}$$

which can be reformulated as

$$\begin{array}{lll} \max & f(x_{\bar{K}}, x_{K_f}, e_{\overline{K_f}}) & \\ s.t. & (x^s_{\bar{K}}, x^s_{K_s}) \in X'_s & \forall s \in S \end{array}$$

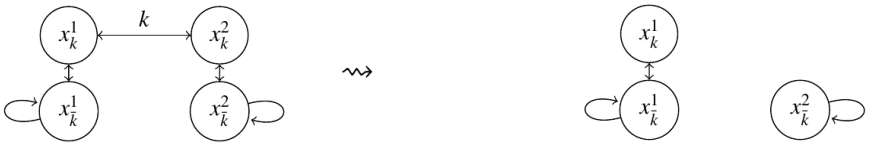


Figure 3.2: Graph representation of the modification yielding the *restrict split* variant.

where $X'_s = X^s \cap \bigcap_{k \in K_s} X'_k$, and X'_k is a set such that $x^s_k \in X'_k$ if and only if $(x^s_k, e^{\bar{s}}_k) \in X^k$. In this restriction of the problem the constraints indexed by k are not coupling constraints anymore, because there is a single subsystem to which variables remaining free in the coupling constraint belong. Thus, because that S is a partition of N, we have that $X' = \prod_{s \in S} X'^s$. Finally, assuming that $f(x)$ is separable along S, Corollary C.1 implies that this restriction of the problem is decomposable.

Example 3.2. *The following illustrates the* restrict split *uncoupled variant of the original problem, obtained by fixing variables from subsystems other than s_1 which also appear in coupling constraint k.*

3.3 Decomposable relaxations and upper bound sets

As specified by Definition 3.6, a multiobjective relaxation can be obtained either by replacing some objective functions f_j by "relaxed" functions f'_j or by considering a superset X' of the original feasible set X. We focus here on the latter possibility, which is to be achieved by relaxing the constraints defining X. The simplest way to achieve this is to simply ignore these constraints. The relaxations we wish to consider must also allow us to reach decomposability. For this purpose we need to ignore, in these constraints, only *terms* which are associated with *some* coupling variables, while ensuring that the resulting feasible set X' is a superset of X.

Consider $N = \{1, ..., n\}$ a set of variable indices, $b \in \mathbb{R}$, $g(x_i \mid i \in N)$ the function defining the left-handside of a coupling constraint, and feasible set $X = \{x \in \mathbb{R}^n \mid g(x_i \mid i \in N) \leqslant b\}$. Further consider $I \subseteq N$. We denote $(x_i \mid i \in N)$ by x, $(x_i \mid i \in I)$ by x_I and $(x_i \mid i \in N \backslash I)$ by $x_{\bar{I}}$. We say that g is *relaxable* for I if and only if there exists some function $g'(x_I)$ such that $g'(x_I) \leqslant g(x)$ for all x in the domain of g. If g' relaxes g for $I \subseteq N$, then for any $b' \in \mathbb{R}$ such that $b' \geqslant b$, $X' = \{x \in \mathbb{R}^n \mid g'(x_I) \leqslant b'\} \supseteq X$.

Linear budget constraints (with positive coefficients) are common relaxable constraints: omitting any variable yields a relaxation of the original constraint.

Example 3.3. *In the case of the knapsack problem, where N is a set of items, w_i is the weight of item $i \in N$ and b is the capacity, the budget constraint $g(x) = \sum_{i \in N} w_i x_i \leqslant b$ is relaxable for any subset I of objects. The relaxing constraint is that obtained by omitting all variables $x_{\bar{I}}$, so $g'(x_I) = \sum_{i \in I} w_i x_i$.*

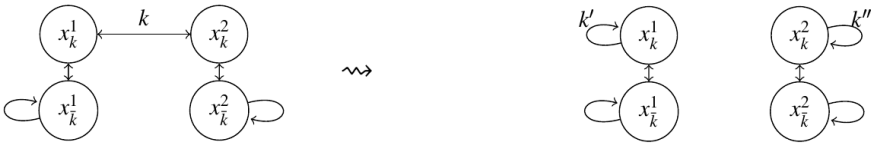


Figure 3.3: Graph representation of the modification yielding the *copy split* variant.

3.3.1 Copy-splitting coupling constraints

For each $k \in K$, let $\gamma(k)$ be the set of subsystems which have variables appearing in constraint k. We consider a variant of the original problem obtained by producing as many relaxations of each coupling constraint k, as there are subsystems in $\gamma(k)$. Each relaxation involves only variables from one of these subsystems.

Formally, assuming that coupling constraint k is relaxable for each $s \in \gamma(k)$, we denote by $g_k'^s(x_k^s) \leqslant b_k$ the relaxation of $g_k(x_k) \leqslant b_k$ associated with $s \in |\gamma(k)|$. Then the following problem is a relaxation of problem (P), in the sense that its feasible set X' is such that $X \subseteq X'$.

$$\begin{array}{lll} \max & f(x) & \\ s.t. & g_k'^s(x_k^s) \leqslant b_k & \forall\, k \in K, s \in \gamma(k) \\ & x^s \in X^s & \forall\, s \in S \end{array}$$

Setting $X'^s = \{x \in X^s \mid g_k^s(x^s) \leqslant b_k\ \forall k \in K, s \in \gamma(k)\}$ the relaxation can be rewritten as:

$$\begin{array}{ll} \max & f(x) \\ s.t. & x^s \in X'^s \end{array}$$

Thus, we have $X' = \prod_{s\in S} X'^s$. Then, under the assumption that f is separable along S, Corollary C.1 implies that this relaxation is decomposable along S.

Example 3.4. *The following illustrates the uncoupled variant of the original problem obtained by replacing coupling constraint k by two constraints k' and k'', which are copies of k which have been restricted to variables of each of the subsystems.*

3.3.2 Local relaxation of coupled variables in subproblem constraints

In the next decouplable variant of the original problem (P), we assume that each constraint $x^s \in X^s$ is of the form $g^s(x^s) \leqslant b^s$ and is relaxable for $s \cap \bigcup_{k\in K} k$. In other words, for $\bar{K}$ the set of non-coupled variables, $g^s(x^s) \leqslant b^s$ is relaxed into $g'^s(x^s_{\bar{K}}) \leqslant b^s$. Applying this relaxation to all $s \in S$, we obtain the following relaxation:

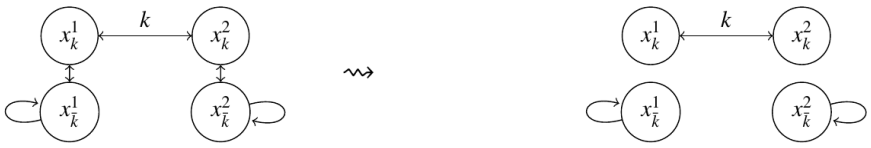


Figure 3.4: Graph representation of the modification yielding the *local relaxation* variant.

$$\begin{array}{lll} \max & f(x) & \\ s.t. & x^k \in X^k & \forall\, k \in K \\ & g'^s(x^s_{\bar{K}}) \leqslant b^s & \forall\, s \in S \end{array}$$

Although the constraints for the subproblems now involve only strictly local variables, coupled variables still appear free in the objective function, and their assignments are still constrained by the coupling constraint, so the following formulation is a relaxation of the original problem.

The set

$$\left\{ \begin{array}{lll} K, & & \text{(variables appearing in coupling constraints)} \\ \bar{K}^s & \forall s \in S & \text{(uncoupled variables local to subsystem } s) \end{array} \right\}$$

defines a partition of N, with each set of constraints involving variables from one element of the partition, and thus

$$X' = X_K \times \prod_{s \in S} X^s_{\bar{K}}$$

Assume that for all $j \in \{1, ..., p\}$, f_j is separable along $\{K, \bar{K}\}$, and that the term associated with $\bar{K}$ can be further separated along S, i.e.

$$f_j(x) = \quad f_j^K(x_K) + \sum_{s \in S} f^s_{j,\bar{K}}(x^s_{\bar{K}})$$

Then, Corollary C.1 implies that this variant is decomposable along S and along $\{K, \bar{K}\}$.

Example 3.5. *Figure C.8 illustrates the uncoupled variant of the original problem obtained by removing terms associated with coupled variables from non-coupling constraints in subsystem local constraints.*

3.4 Experimental results

In this section, we report the results of an experiment serving two purposes. First, to report the computation times for our lower bound (LB) and upper bound (UB) concepts, and compare theses results with the computation time of the actual non-dominated set of the problem. Second, to evaluate how well they approximate this non-dominated set. We make these measurements as functions of the number of objectives (column p), the number of variables in the instance (column n), the number of subsystems in which the instance is subdivided (column ($|S|$), the number of coupling constraints in the instance (column $|K|$), and the proportion of variables in each subproblem that are coupled by each coupling constraint (column DV). We assume that when a constraint is coupling, it couples variables from all systems, at the rate defined by DV.

Experiments are conducted solving instances of the following *generic coupled problem* (GCP), which is formulated as:

$$
\begin{array}{lll}
\max & f(x) = (\sum_{i\in N} \pi_i^1 x_i, ..., \sum_{i\in N} \pi_i^p x_i) = (\sum_{s\in S}\sum_{i\in s} \pi_i^1 x_i, ..., \sum_{s\in S}\sum_{i\in s} \pi_i^p x_i) & \\
s.t. & \sum_{s\in S}\sum_{i\in s\cap k} a_i x_i \leqslant b_k & \forall\, k \in K \\
& \sum_{i\in s} c_i x_i \leqslant d_s & \forall\, s \in S \\
& x_i \in \{0,1\} & \forall\, i \in N
\end{array}
\qquad (GCP)
$$

With $f(x) : \{0,1\}^n \to \mathbb{N}^p$. Because all constraints in the problem are budget constraints, for all $i \in N$, 0 is a *neutral* value for x_i, which can then be used in all restrictions.

For every $i \in N$, we randomly choose $\pi_i^j, a_i, c_i, \in [1, 1000]$. For each $k \in K$, $b_k = \left\lceil \frac{\sum_{s\in S}\sum_{i\in s\cap k} a_i}{2} \right\rceil$, and for each $s \in S$, $d_s = \left\lceil \frac{\sum_{i\in S} c_i}{2} \right\rceil$. For each subsytem $s \in S$, we choose $\left\lceil \frac{DV\times|s|}{100} \right\rceil$ variables to appear in coupling constraint $k \in K$, where $|s|$ is the number of variables involved in subsystem s.

When $p = 2$, non decomposed original problems and subsystem problems are solved using a straightforward implementation of the bi-objective e-constraint method: while the problem is feasible we optimize lexicographically the first, then the second objective, under the constraint that each new solution is better than the previous one on the second objective. This is achieved by simply updating the lower bound of the same constraint. For $p \geqslant 3$, we use the generic MOCO algorithm by Tamby and Vanderpooten (2021). Once solutions to subsystem problems have been obtained, they are *pooled* together according to Algorithm 6. At each step of the pooling process, some explicit dominance filtering needs to be performed. Figure C.9 exhibits the enframing of the original non-dominated set between the various notions of bounds presented in this article.

First, we study the increase in performance yielded by decomposition alone, in the case where the original optimization problem is uncoupled. In Table 3.1, we report the results of an experiment where we solved uncoupled instances without decomposition, and then using decomposition. Computing time for each of these operations is respectively denoted as $T.NoDec$, and $T.Dec$. Val-

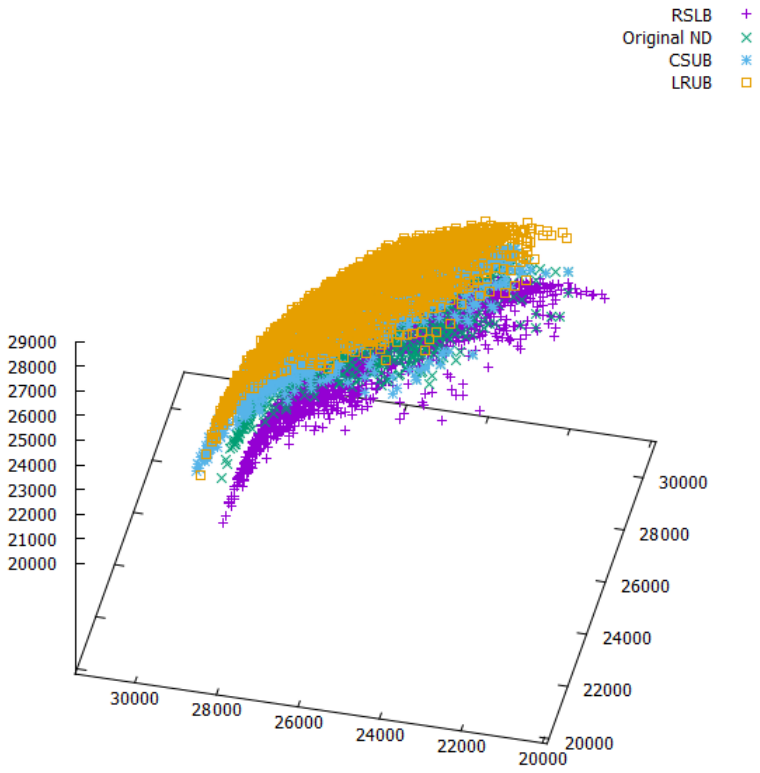


Figure 3.5: Plot representation of the original non-dominated set and bound sets. Instance parameters are $p = 3$, $n = 75$, $|S| = 2$, $|K| = 1$, DV = 10

p	n	$\|S\|$	$\|ND\|$		$T.NoDec$		$T.Dec$		$\frac{T.Dec}{T.NoDec}(\%)$	
2	150	3	239.80	[67.28]	28.82	[20.71]	5.11	[2.12]	17.71	[10.24]
2	200	3	422.00	[71.49]	54.62	[21.05]	7.60	[1.44]	13.91	[6.82]
3	75	3	1339.70	[402.18]	259.26	[103.79]	14.50	[4.13]	5.59	[3.98]
3	100	3	2964.20	[1190.81]	1057.01	[554.71]	33.43	[12.93]	3.16	[2.33]
4	50	3	452.70	[319.79]	1486.30	[865.19]	14.22	[6.33]	0.96	[0.73]
4	60	3	2296.18	[1377.80]	4001.50	[1321.01]	38.45	[14.39]	0.96	[1.09]

Table 3.1: Effect of decomposition on the resolution of an uncoupled problem. Average values for 10 trials.

ues reported between squared brackets are standard deviations. We found that decomposition yields an improvement in computing time which increases with the number of criteria and with the number of variables when $p = 2, 3$. For $p = 4$, $n = 60$ and $|S| = 3$, resolution using decomposition took, on average, less than 1% of the time needed to solve the problem without decomposition.

We then consider uncoupled restrictions and relaxations of coupled instances. Solving a *restrict split* restriction of (GCP) requires us to produce some assignment of couplings to subsystems. For any $k \in K$, we need to select a subsystem $s \in \gamma(k)$. The selection is made according to the following heuristic:

$$s_k = \arg\max_{s \in \gamma(k)} \sum_{i \in k \cap s} \frac{\sum_{j=1}^{p} \pi_i^j}{c_i}$$

In other words, we choose the subsystem coupled by k such that the variables of this subsystem which are coupled by coupling k maximize the average sum of profit-cost ratios over criteria.

Relaxed constraints for the *copy split* variant are obtained as follows. For any $k \in K$, if $\sum_{s \in S} \sum_{i \in s \cap k} a_i x_i \leqslant b_k$, then because coefficients a_i are positive, for any $s \in S$, we have $\sum_{i \in s \cap k} a_i x_i \leqslant b_k$, so that for any $k \in K$ and $s \in S$, the latter constraint is a relaxation of the former. The copy split relaxation will thus be:

$$\begin{array}{lll} \max & f(x) & \\ s.t. & \displaystyle\sum_{i \in s \cap k} a_i x_i \leqslant b_k & \forall\, s \in S, \forall k \in K \\ & \displaystyle\sum_{i \in s} c_i x_i \leqslant d_s & \forall\, s \in S \\ & x_i \in \{0,1\} & \forall\, i \in N \end{array} \qquad \text{(CS GCP)}$$

Relaxed constraints for the *locally relaxed* variant are obtained by considering that, because coefficients c_i are positive, we have $\sum_{i \in s \cap \bar{K}} c_i x_i \leqslant \sum_{i \in s} c_i x_i$, where $s \cap \bar{K}$ stands for $s \setminus \bigcup_{k \in K} k$. Thus $\sum_{i \in s \cap \bar{K}} c_i x_i \leqslant d_s$ is a relaxation of $\sum_{i \in s} c_i x_i \leqslant d_s$, which will be substituted to the latter in the locally relaxed variant. The locally relaxed variant will thus be:

$$\begin{array}{lll} \max & f(x) & \\ s.t. & \displaystyle\sum_{s \in S} \sum_{i \in s \cap k} a_i x_i \leqslant b_k & \forall\, k \in K \\ & \displaystyle\sum_{i \in s \cap \bar{K}} c_i x_i \leqslant d_s & \forall\, s \in S \\ & x_i \in \{0,1\} & \forall\, i \in N \end{array} \qquad \text{(LR GCP)}$$

To measure the quality of the approximation provided by a bound set, we use *a posteriori* ε-dominance, defined as follows :

Definition 3.9. *Given $y, y' \in Y$, y is said to ε-dominate y', written $y \succeq_\varepsilon y'$ if and only if*

$$y_j(1+\varepsilon) \geqslant y'_j \quad \forall j \in \{1, ..., p\}$$

Definition 3.10. *A is an ε-approximation of B for $\succeq_\varepsilon$ if and only if*

$$\forall y \in B \; \exists y' \in A, y' \succeq_\varepsilon y$$

Here, we use this notion to evaluate the quality of approximation of a set A by a set B *a posteriori*. This is achieved by computing the smallest ε such that, for all $y \in A$, there is a point in the approximation B set which ε-dominates it. Formally, we compute

$$\varepsilon^*(A, B) := \min\{\varepsilon \in \mathbb{R} \mid \forall y' \in A \; \exists y \in B, y' \succeq_\varepsilon y\}$$

To measure how well a lower bound set L on a set Y approximates Y, we compute $\varepsilon^*(Y, L)$. To measure how well an upper bound set U to a set Y approximates Y, we compute $\varepsilon^*(U, Y)$. Another way to measure approximation quality is to measure the actual proportion of the non-dominated set of the original problem which is already contained in the bound set, which we will denote, e.g. in the case of an LB set, by $\frac{|ND \cap LB|}{|ND|}$.

In the following tables, *RS* denotes the *restrict split* restriction (Section 3.2.2) of the original problem, which provides a lower bound set. *CS* denotes the *copy split* relaxation (Section C.3.3.1) of the problem, which provides an upper bound set. *LR* denotes the relaxation obtained by omitting coupled variables in local subproblem constraints (Section C.3.3.2), which also provides an upper bound set. $T.$ denotes computing time, $|.|$ denotes the size of a set of points, and $ApQ.(\varepsilon)$ denotes the quality of approximation in terms of *a posteriori* ε dominance.

Let us begin by observing that in the bi-objective case, computing the bounds may not be relevant, because the whole non-dominated set is computed quickly. For more than two objectives however, as $T.ND$ increases strongly, computing these bounds becomes worthwhile.

In Table 3.2, we report the average computing time $T.ND$ and size of the non-dominated sets of the instances for which we computed bound sets, as well as their number of variables. This will serve as a reference to measure the advantages of computing bound sets using decomposition, rather than computing the original non-dominated sets.

From Table 3.3, we can observe the impact of varying instance parameter values on the time required to compute the RS lower bound. Increasing the number of subsytems while keeping n fixed makes the computation of the bound significantly faster, as the problem is divided into smaller subproblems, with an RS lower bound obtained in less than 2% of the time required to obtain the original non-dominated set when $|S| = 4$. An increase in the number of coupling constraints $|K|$ or in the proportion of coupling variables also speeds up the computation of the lower bound, because the greater the proportion of coupled variables in a problem, the smaller the number of variables in the *restrict split* variant.

p	n	$\|S\|$	$\|K\|$	DV	$T.ND$		$\|ND\|$	
2	200	2	1	10	49.99	[13.16]	426.80	[54.63]
2	200	3	1	10	59.27	[14.79]	382.60	[39.54]
2	200	4	1	10	70.87	[23.50]	419.00	[65.95]
2	200	2	2	10	90.80	[41.41]	420.20	[69.15]
2	200	2	3	10	89.24	[20.39]	427.04	[59.82]
2	200	2	1	20	82.04	[24.73]	450.50	[76.25]
3	100	2	1	10	1663.06	[502.95]	3452.58	[577.36]
3	100	3	1	10	933.32	[461.91]	2601.62	[886.65]
3	100	4	1	10	859.59	[262.40]	2751.31	[556.69]
3	100	2	2	10	1647.33	[1042.60]	3179.33	[1281.37]
3	100	2	3	10	2727.04	[1192.15]	4372.53	[893.25]
3	100	2	1	20	1971.90	[866.14]	3363.50	[997.37]
4	60	2	1	10	2304.04	[1740.95]	3836.13	[1421.05]
4	60	3	1	10	1908.62	[1735.73]	3858.21	[2565.50]
4	60	2	2	10	2334.97	[1276.06]	3929.67	[1650.62]

Table 3.2: Computation of the set of non-dominated points for the original reference instance. Average values for 10 trials.

p	n	$\|S\|$	$\|K\|$	DV	$T.RS$		$\frac{T.RS}{T.ND}(\%)$		$\|RS\|$		$ApQ.(\varepsilon)$		$\frac{\|ND \cap RS\|}{\|ND\|}$	
2	200	2	1	10	12.62	[2.33]	25.25	[17.71]	393.82	[62.68]	0.02	[0.01]	0.02	[0.06]
2	200	3	1	10	6.69	[0.66]	11.29	[4.46]	361.72	[35.38]	0.03	[0.01]	0	[0]
2	200	4	1	10	6.64	[1.69]	9.37	[7.19]	394.68	[70.39]	0.03	[0.01]	0	[0]
2	200	2	2	10	13.40	[2.21]	14.76	[5.34]	364.44	[73.81]	0.04	[0.01]	0	[0]
2	200	2	3	10	15.64	[3.38]	17.53	[16.56]	360.03	[57.99]	0.06	[0.02]	0	[0]
2	200	2	1	20	13.91	[4.39]	16.96	[17.75]	394.15	[66.44]	0.05	[0.01]	0	[0]
3	100	2	1	10	92.68	[23.08]	5.57	[4.59]	3148.90	[645.92]	0.02	[0.01]	0.26	[0.26]
3	100	3	1	10	29.49	[15.74]	3.16	[3.41]	2258.62	[842.79]	0.03	[0.01]	0.24	[0.31]
3	100	4	1	10	16.77	[5.52]	1.95	[2.10]	2559.37	[627.78]	0.03	[0.01]	0.17	[0.30]
3	100	2	2	10	73.80	[28.27]	4.48	[2.71]	2703.27	[1086.52	0.05	[0.02]	0	[0]
3	100	2	3	10	112.53	[51.92]	4.13	[4.36]	3350.13	[965.83]	0.06	[0.02]	0	[0]
3	100	2	1	20	68.26	[20.07]	3.46	[2.32]	2492.41	[770.68]	0.05	[0.02]	0	[0]
4	60	2	1	10	128.87	[58.60]	5.59	[3.37]	3910.98	[1697.70]	0.03	[0.02]	0.31	[0.36]
4	60	3	1	10	30.29	[17.59]	1.59	[1.01]	3222.32	[1938.03]	0.03	[0.02]	0.25	[0.28]
4	60	2	2	10	102.67	[61.53]	4.40	[4.82]	2456.53	[648.04]	0.04	[0.02]	0.18	[0.25]

Table 3.3: Computation of the lower bound set associated with the *restrict split* restriction (RS lower bound). Average values for 10 trials.

p	n	$\|S\|$	$\|K\|$	DV	$T.CS$		$\frac{T.CS}{T.ND}(\%)$		$\|CS\|$		$ApQ.(\varepsilon)$		$\frac{\|ND \cap CS\|}{\|ND\|}$	
2	200	2	1	10	14.13	[1.74]	28.27	[13.22]	439.90	[62.97]	0.01	[0]	0.28	[0.34]
2	200	3	1	10	7.52	[0.73]	12.69	[4.94]	396.63	[35]	0.01	[0]	0.21	[0.35]
2	200	4	1	10	6.81	[1.58]	9.61	[6.72]	400.49	[78.36]	0.01	[0]	0.17	[0.22]
2	200	2	2	10	17.31	[2.63]	19.06	[6.35]	444.11	[57.59]	0.01	[0]	0	[0.01]
2	200	2	3	10	23.31	[4.99]	26.12	[24.45]	456.12	[60.25]	0.01	[0.01]	0	[0.00]
2	200	2	1	20	16.85	[4.46]	20.54	[18.03]	453.32	[83.70]	0.01	[0.01]	0.03	[0.07]
3	100	2	1	10	104.62	[21.90]	6.29	[4.35]	3741.15	[694.08]	0.01	[0.01]	0.35	[0.34]
3	100	3	1	10	34.49	[14.35]	3.69	[3.11]	2930.96	[924.46]	0.01	[0.01]	0.29	[0.23]
3	100	4	1	10	20.03	[5.14]	2.33	[1.96]	2888.02	[538.25]	0.01	[0.01]	0.47	[0.37]
3	100	2	2	10	107.08	[38.20]	6.50	[3.66]	3639.71	[1304.65]	0.02	[0.01]	0.06	[0.07]
3	100	2	3	10	146.64	[43.17]	5.38	[3.62]	4588.33	[968.82]	0.02	[0.01]	0.09	[0.16]
3	100	2	1	20	113.28	[46.60]	5.74	[5.38]	3842.65	[1237.79]	0.01	[0.01]	0.26	[0.20]
4	60	2	1	10	126.15	[45.67]	5.48	[2.62]	3913.12	[1385.05]	0.02	[0.01]	0.64	[0.38]
4	60	3	1	10	33.85	[19.53]	1.77	[1.13]	4126.19	[2848.88]	0.01	[0.01]	0.75	[0.28]
4	60	2	2	10	154.95	[66.09]	6.64	[5.18]	3924.00	[1480.57]	0.03	[0.01]	0.28	[0.36]

Table 3.4: Computation of the upper bound set associated with the *copy split* relaxation (CS upper bound). Average values for 10 trials.

Approximation quality appears to be very good when $|K| = 1$ and $DV = 10$, with between 17 and 26% of non-dominated points contained in the RS lower bound when $p = 3$ and between 27 and 51% when $p = 4$. A slight deterioration in approximation quality is observed as $|K|$ and DV increase, which is more readily explained by the neutralization of a higher number of variables.

Now we turn to experiments with upper bounds. Results for the CS upper bound are reported in Table 3.4. The computation of this bound appears to require more time than does the RS lower bound. However, it benefits very similarly from an increase in n and in $|S|$, with a CS upper bound obtained in 1.2% of the time required to compute the original non-dominated set, for $p = 3$ and $|S| = 4$. An increase in $|K|$ or in DV does not appear to have a significant impact, however, on computation time for the CS upper bound.

We observe that this upper bound consistently provides an excellent approximation quality of 1% when $|K| = 1$ and $DV = 10$, with on average between 34% and 47% of the non-dominated points already contained in the CS upper bound for $p = 3$, and between 41% and 54% when $p = 4$. Approximation quality decreases but remains good when $|K|$ or DV increase.

Finally, turning to Table 3.5, we can observe that the LR upper bound is obtained strikingly faster than the set of non-dominated points: it required less than 1 % of the time in all our experiments. This most likely illustrates the power of dividing the original problem into a larger number of subsystems. Indeed in this case, instead of being associated with one of the $|S|$ subsystems, the coupled variables constitute a separate, small subsystem. The impact of increasing n is similar to what was previously observed. An increase in $|K|$ appears to have a detrimental effect on computing time for the LR upper bound, while an increase in DV has an unclear effect. Surprisingly, an increase in $|S|$ does not decrease the ratio between the computing time of the LR upper bound and that of the original non-dominated set. Indeed, it seems that increasing $|S|$ also makes the global resolution of the original problem faster, albeit not to the same proportion as

p	n	$\|S\|$	$\|K\|$	DV	$T.LR$		$\frac{T.LR}{T.ND}(\%)$		$\|LR\|$		$ApQ.(\varepsilon)$	
2	200	2	1	10	11.08	[1.73]	22.16	[13.15]	426.44	[61.71]	0.05	[0.01]
2	200	3	1	10	6.54	[0.89]	11.03	[6.02]	416.20	[51.26]	0.04	[0.01]
2	200	4	1	10	6.56	[2.23]	9.26	[9.49]	452.54	[88.52]	0.04	[0.01]
2	200	2	2	10	11.01	[2.14]	12.13	[5.17]	421.10	[85.76]	0.09	[0.01]
2	200	2	3	10	10.82	[2.57]	12.12	[12.61]	351.91	[53.51]	0.12	[0.02]
2	200	2	1	20	9.62	[2.77]	11.73	[11.20]	419.91	[94.16]	0.09	[0.02]
3	100	2	1	10	1.21	[0.40]	0.07	[0.08]	3354.34	[787.92]	0.06	[0.01]
3	100	3	1	10	0.97	[0.49]	0.10	[0.10]	2690.81	[1067.95]	0.04	[0.01]
3	100	4	1	10	0.94	[0.35]	0.11	[0.13]	2749.46	[776.61]	0.05	[0.01]
3	100	2	2	10	1.84	[0.63]	0.11	[0.06]	3251.36	[1013.56]	0.09	[0.02]
3	100	2	3	10	2.37	[1.13]	0.09	[0.09]	3355.25	[1065.60]	0.14	[0.01]
3	100	2	1	20	2.33	[1.08]	0.12	[0.12]	2932.84	[1195.43]	0.11	[0.02]
4	60	2	1	10	1.88	[1.37]	0.08	[0.08]	4128.46	[1761.28]	0.05	[0.01]
4	60	3	1	10	1.80	[1.52]	0.09	[0.09]	3720.42	[2406.99]	0.06	[0.02]
4	60	2	2	10	1.19	[0.57]	0.05	[0.04]	2920.16	[1112.89]	0.09	[0.02]

Table 3.5: Computation of the upper bound set associated with the *locally relaxed* relaxation (LR upper bound). Average values for 10 trials.

decomposition. However in the case of the LR upper bound, it appears that this effect was not as beneficial to computing the LR upper bound using decomposition.

On the downside, the LR upper bound provides a way poorer upper approximation of the non-dominated set than the CS upper bound does. In all our experiments, the proportion of points from the original non-dominated set in the LR upper bound was 0, and the a posteriori value of ε was more than twice and up to ten times larger than for the CS upper bound.

In Appendix A, we propose the definition of a solution concept base on computing a lower bound and an upper bound on the non-dominated set of a MO problem. This double bound can either be used on it's own as an approximation of the non-dominated set of the original problem. The accuracy of this approximation can be defined either by ε-approximation, or by the difference in hypervolume dominated by the bounds. Alternatively, this double bound can be used as an element in a wider decision making process, providing information regarding the levels of performance that can be reached on each criterion. In the next chapter, we introduce an application problem that is a coupled problem, as well as the theoretical notions which are relevant to the study of its structure.

3.5 Conclusions and discussion

In this chapter, we have defined a collection of restrictions and relaxations of a coupled problem, which provide upper and lower bound sets to the non-dominated set of a coupled problem. The restrictions and relaxations were designed to be decomposable, and solved by solving independent subproblems and pooling the results.

On the one hand, we found that the *restrict split* and *copy split* bound provide very good

approximations of the non-dominated set. The availability of decomposition to compute them provides a sizable advantage in terms of computation time, which increases as the number of variables and the number of subsystems increases. However, it may be argued that these bounds are still too expensive to compute.

On the other hand, we found that the *locally relaxed* upper bound can be obtained dramatically faster than the original non-dominated set, because it not only decomposes the problem along subsystems lines but also separates the coupled part of the problem from the rest. However, it provides a poorer approximation of the non-dominated set. A natural development of the work presented in this paper is to find ways to improve the quality of this upper bound concept while retaining the structural property that makes its computation very fast.

Part of the cost of computing bound sets comes from the sheer number of non-dominated points to be found, even for subproblems. When the goal is to produce an approximation of the non-dominated set of a problem, generating all solutions of the restriction or relaxation may not be useful. A priori ε-approximation (see e.g. Vassilvitskii and Yannakakis (2005), Bazgan et al. (2015)) could generate approximations of smaller size, but because it may contain dominated points, it cannot be used to approximate *upper* bound sets. Methods developed by Kaddani et al. (2017) can generate a subset of the non-dominated set of any problem, but to do so, it requires additional preference information from the decision maker. Knowing these challenges, further work should pursue the generation of subsets of decomposable upper bound sets.

Chapter 4

Application Problem [1]

Chapter Abstract

In this chapter, we will study an application problem called *REF*, a particular case of the *generic coupled problem*. REF is a multi-site assignment problem with coupling constraints, and admits a dynamic programming resolution algorithm. We apply the various notions of uncoupled lower and upper bounds to REF, and provide dynamic programming algorithms to compute them. We show that decomposition as well as other computational tricks can improve the initial dynamic programming approach. Facing some intrinsic limitations of the initial "bottom-up" approach, we switch the perspective and use dynamic programming as a subprocedure in a "top-down" approach which tries to enumerate, as efficiently as possible, all ways of decoupling the problem.

Contents

4.1 Presentation of the REF problem 78
4.1.1 Formulation 78
4.1.2 Relation to other problems and degenerate variants 80
4.1.3 A basic dynamic programming resolution method 82
4.2 Upper and lower bound sets for REF 87
4.2.1 Weak upper bound sets and a strong singleton upper bound 87
4.2.2 Strong and weak lower bound sets 92
4.2.3 Experimental Results 97
4.3 "Bottom-up" approaches based on Dynamic Programming 98
4.3.1 Filtering with a set of incumbent solutions 99
4.3.2 Partial filtering 99
4.3.3 "Kickstarting" 101
4.3.4 Experimental Results 102
4.4 "Top-down" approaches for solving REF 105
4.4.1 Enumerating all ways of decoupling REF 106
4.4.2 Pre-computation of independent dynamic programming states 108
4.4.3 Towards a branch and bound method for solving coupled problems . . . 111
4.5 Conclusions and perspectives 120

[1] Earlier versions of this chapter were presented orally at the RAMOO 2018 and ROADEF 2019 conferences.

4.1 Presentation of the REF problem

Original concepts developed in this work are intended to be applied to the resolution of a particular application problem, which we will refer to as *REF*. REF is an assignment problem where, to use classical terminology, *tasks* have to be assigned to *machines* in order to maximize some profit value associated with performing each task on some machine. Furthermore, machines are to be thought of as located on *sites*, with each site having limited *capacity*, which is consumed by assignments of tasks to machines located on it. The decision maker assumes the position of a central planner, who has to allocate tasks to machines on the different sites.

4.1.1 Formulation

REF can be dressed up as a *multi-site production problem*. Let T be a set of *tasks*, S a set of *sites* and M a set of *machines*. We will say that $S(t) \subset S$ is the set of sites on which task t can be performed. Symmetrically $T(s) \subseteq T$ will be the set of tasks that can be performed on site $s \in S$. We assume that $t \in T$ can in general be assigned to any machine of a site it can be performed on. Therefore, given $M(s) \subset M$ the set of machines located on site s, we will denote by $M(S(t))$ the set of machines on which task t can be performed, i.e. for all $t \in T, M(S(t)) = \bigcup_{s \in S(t)} M(s)$. Finally, $N := \{(t,m) \in T \times M \mid m \in M(S(t))\}$ will denote the set of all variable indices, with $|N| =: n$.

Each assignment of a task t to a machine m yields profits g^j_{tm}, for $j \in \{1, ..., p\}$, and consumes an amount w_{tm} of resources available on the unique site $s \in S$ such that $m \in M(s)$. This resource should be thought of as time or energy available to a manufacturing site. Each of the p objectives is to be maximized under the constraints that each task should be assigned at most once, and that for each site, resource consumption does not exceed some bound. Example C.5 provides a graph illustration of the assignment problem and examples of feasible and unfeasible solutions.

Example 4.1. *Consider and instance of REF defined by* $T = \{t_1, t_2, t_3\}$, $M = \{m_1, m_2, m_3, m_4\}$ *and* $S = \{s_1, s_2\}$, *such that* $S(t_1) = \{s_1, s_2\}$, $S(t_2) = \{s_1\}$, $S(t_3) = \{s_2\}$, $M(s_1) = \{m_1, m_2\}$, $M(s_2) = \{m_3, m_4\}$. *The following table records the weights associated with each assignment of a task to a machine. Assume that the capacity of each site is* 1.

	w_{tm}	t_1	t_2	t_3
s_1	m_1	0.6	0.5	
	m_2	0.5	0.5	
s_2	m_3	0.6		0.5
	m_4	0.5		0.6

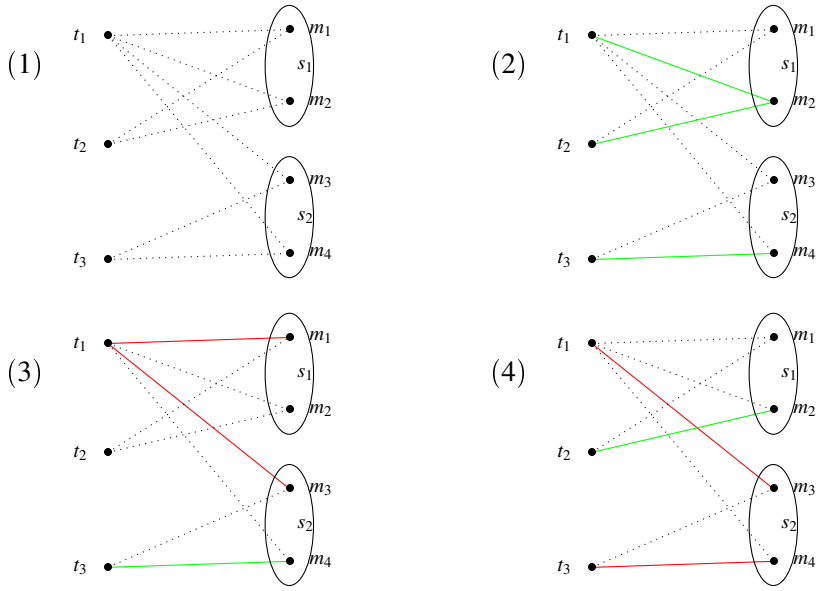


Figure 4.1: Graph (1) provides a representation of all possible assignments, as well as the grouping of machines m_1 and m_2 on site s_1, and machines m_3 and m_4 on site s_2. Task t_1 can be performed either on s_1 or on s_2, task t_2 is local to site s_1, and task t_3 is local to site s_2. Graph (2) shows a feasible solution, and Graph (3) and (4) show two unfeasible solutions. Solution (3) is unfeasible because task t_1 is assigned twice. Solution (4) in unfeasible because the sum of weights of assignments to machines of site s_2 is 1.1, which is larger than 1.

The REF problem can be written as the following 0-1 linear program.

$$
\begin{aligned}
\max \quad & \sum_{t \in T} \sum_{m \in M(S(t))} g^j_{tm} x_{tm} && \forall\, j \in \{1, ..., p\} \\
s.t. \quad & \sum_{m \in M(S(t))} x_{tm} \leqslant 1 && \forall\, t \in T \\
& \sum_{t \in T(s)} \sum_{m \in M(S(t))} w_{tm} x_{tm} \leqslant b_s && \forall\, s \in S \\
& x_{tm} \in \{0, 1\} && \forall\, (t, m) \in N
\end{aligned}
$$

A feature of this problem which captures our interest is that for some $t \in T$, $|S(t)| = 1$, i.e. these tasks can be performed on one site only. We refer to the set of such tasks as T_L, and to $T \backslash T_L$ as T_C. We will often call T_L the set of *local tasks*, and T_C the set of *complex tasks*. We will denote by $T_L(s)$ the set $T_L \cap T(s)$ of local tasks which can be performed on site s. These observations allow the following reformulation:

$$
\begin{aligned}
\max \quad & \sum_{s\in S}\sum_{t\in T(s)}\sum_{m\in M(S(t))} g^j_{tm}x_{tm} && \forall\, j\in\{1,...,p\} && \\
s.t. \quad & \sum_{m\in M(S(t))} x_{tm} \leqslant 1 && \forall\, t\in T_C && (1)\\
& \sum_{m\in M(S(t))} x_{tm} \leqslant 1 && \forall\, s\in S, \forall\, t\in T_L(s) && (2)\\
& \sum_{t\in T(s)}\sum_{m\in M(S(t))} w_{tm}x_{tm} \leqslant b_s && \forall\, s\in S && (3)\\
& x_{tm}\in\{0,1\} && \forall\,(t,m)\in N &&
\end{aligned}
$$

REF is a coupled system MOCO, for which the set of *decisions* is $N = \{(t,m) \mid t \in T, m \in M(S(t))\}$ which corresponds to the set of indices of decision variables. The set (1) of inequalities describe the coupling constraints, indexed by T_C, which corresponds to the set of couplings K in problems formulated in Section C.1.2.1. S naturally corresponds to the set of subsystems. Thus we will say that subsystems or *sites* s and s' are coupled if there is a task $t \in T_C$ such that $(s,s') \in S(t)^2$. Sets of inequalities (2) and (3) represent local constraints, which involve only assignments of tasks which can be performed on one site only, and limit the consumption of resources on each site.

4.1.2 Relation to other problems and degenerate variants

4.1.2.1 REF and GAP

At first sight, REF bears a resemblance to the generalized assignment problem (*GAP*). Both problems combine assignment constraints and knapsack constraints. We temporarily focus on the single objective case, so as to use the complexity of single-objective GAP to characterize the complexity of REF. Let us recall the classical formulation of single objective GAP (cf. e.g. Kellerer et al. (2004)):

$$
\begin{aligned}
\max \quad & \sum_{k=1}^{n}\sum_{i=1}^{m} g_{ki}x_{ki} && && \\
& \sum_{i=1}^{m} x_{ki} \leqslant 1 && \forall\, k\in\{1,...,n\} && \\
& \sum_{k=1}^{n} w_{ki}x_{ki} \leqslant b_i && \forall\, i\in\{1,...,m\} && \\
& x_{ki}\in\{0,1\} && \forall\, i\in\{1,...,m\},\ \forall\, k\in\{1,...,n\} &&
\end{aligned}
$$

Closer comparison between GAP and REF reveals the following difference. Consider, as illustrated in Figure 4.2, a bipartite graph with set T, i.e. vertices corresponding to tasks on the lefthandside, and set M, i.e. vertices corresponding to machines on the right hand side. Each

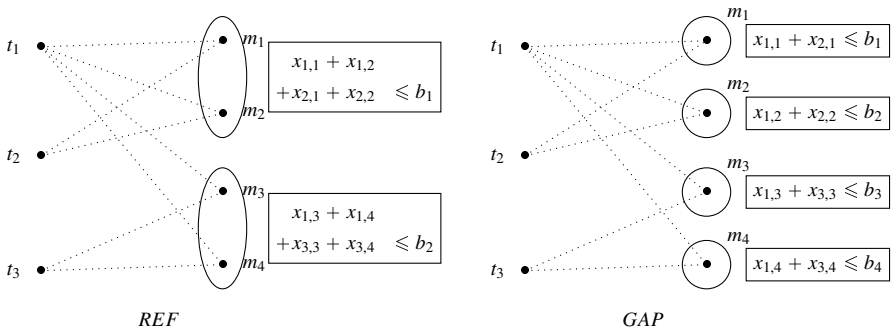


Figure 4.2: Assignment graphs with knapsack constraints for REF (left) and GAP (right). REF has one knapsack constraint per site, while GAP has one knapsack constraint per machine.

edge represents an assignment which is performed by paying some weight. In GAP, a knapsack constraint is associated to each element of the right hand side. However, in REF, a knapsack constraint is associated with a subset of elements of the right hand side, i.e. with $M(s)$, rather than to an element $m \in M$. Still, by degenerating REF to a case where each site has only one machine, we produce instances of GAP, meaning that GAP reduces to REF, which proves at the same time that

Proposition 4.1. *The single objective variant of REF is NP-hard.*

Proof. Assume $|M(s)| = 1$ for each $s \in S$. Then for any $t \in T$, $s \in S(t)$, we denote by x_{ts}, (resp. g_{ts}, w_{ts}) the unique elements of sets $\{x_{tm} \mid m \in M(s)\}$ (resp. $\{g_{tm} \mid m \in M(s)\}$, $\{w_{tm} \mid m \in M(s)\}$). We get the following formulation:

$$
\begin{aligned}
\max \quad & \sum_{t \in T} \sum_{s \in S(t)} g_{ts} x_{ts} & & \\
& \sum_{s \in S(t)} x_{ts} \leqslant 1 & & \forall\, t \in T \\
& \sum_{t \in T(s)} w_{ts} x_{ts} \leqslant b_s & & \forall\, s \in S \\
& x_{ts} \in \{0,1\} & & \forall\, t \in T,\ s \in S
\end{aligned}
$$

which defines an instance of GAP. Thus, GAP reduces to REF, and is therefore NP-hard. □

4.1.2.2 Multiple knapsack variant

Another particular case of REF is obtained when the assignments of some task yields the same profits and consume the same amount of resources for all machines on which the task can be performed. The problem then becomes that of selecting objects endowed with intrinsic value and weight, as in *KP*. Formally, we state that for any $t \in T$, $m, m' \in M(S(t))$, we have $g_{tm} = g_{tm'} =: g_t$ and $w_{tm} = w_{tm'} := w_t$. We then get the following formulation:

$$
\begin{aligned}
\max \quad & \sum_{t\in T}\sum_{s\in S(t)} g_t^j x_{ts} && \forall j \in \{1,...,p\} \\
& \sum_{s\in S(t)} x_{ts} \leqslant 1 && \forall\, t \in T \\
& \sum_{t\in T(s)} w_t x_{ts} \leqslant b_s && \forall\, s \in S \\
& x_{ts} \in \{0,1\} && \forall\, t \in T,\ s \in S
\end{aligned}
$$

Which defines an instance of the *multiple knapsack problem* (MKP, Kellerer et al. (2004)).

4.1.2.3 Multi-Choice Knapsack variant

For any $s \in S$, the problem of assigning tasks $t \in T(s)$ to machines $m \in M(s)$ is a *Multi-Choice Knapsack Problem* (MCKP), formulated as follows :

$$
\begin{aligned}
\max \quad & \sum_{t\in T(s)}\sum_{m\in M(s)} g_{tm}^j x_{tm} && \forall j \in \{1,...,p\} \\
& \sum_{m\in M(S(t))} x_{tm} \leqslant 1 && \forall\, t \in T(s) \\
& \sum_{t\in T(s)}\sum_{m\in M(S(t))} w_{tm} x_{tm} \leqslant b_s && \forall\, s \in S \\
& x_{tm} \in \{0,1\} && \forall\, (t,m) \in N
\end{aligned}
$$

Thus when $T_C = \varnothing$, since for all $t \in T$, and $\{s\} = S(t)$, we have $M(S(t)) = M(s)$, REF can be decomposed into $|S|$ instances of *MCKP*. This variant is of interest because in practice, when we produce an uncoupled variant of an instance of REF, we obtain a collection of such *MCKP* instances.

4.1.3 A basic dynamic programming resolution method

Dynamic programming (DP), as introduced by Bellman (1952), is an algorithmic approach to solving decision problems which can be represented by multi-stage models of some decision process. The key feature of a DP algorithm is to build *partial solutions* incrementally at each decision stage. Stages can be organized in a sequence or a directed states graph (Escoffier and Spanjaard (2005)). At each stage, all previous stages having been performed, a small optimization problem is solved, taking previous stages as input to generate new partial solutions. This optimization eliminates partial solutions which will not be built upon in later stages. On the one hand, this implies that some feasible solutions of the original problem cannot be reached anymore. On the other hand, this is a way to avoid considering all solutions to the original problem, and allows for a quicker resolution. The main challenge of DP is thus to define a sequential decision structure

and intermediary optimization problems which allow the elimination of as many partial solutions as possible, while guaranteeing that efficient solutions remain accessible and are indeed outputted by the algorithm.

For simplicity, we will not define partial solutions on projections of the domain of complete solutions over subsets of variables. Rather we use the fact that 0 is an admissible value for any decision variable of REF. Recall that $N := \{(t,m) \in T \times M \mid m \in M(S(t))\}$, and let $\mathbf{0}_N$ be the solution such that for any $i \in N$, $\mathbf{0}_i = 0$. By definition of an admissible value, $\mathbf{0}_N$ is feasible. We set $Q_0 := \{\mathbf{0}_N\}$ to be the initial stage of the decision process. At each subsequent stage, building solutions according to decisions available at this step will amount to making an assignment of variables which previously took value 0. Thus, solutions generated during the sequential decision process are of equal dimensionality, and we can evaluate them without having to define one objective function per decision stage.

Let $T = \{1, ..., \tilde{t}\}$, the set of tasks in REF, be used as the set of stages of the decision process, where $\tilde{t}$ will denote the last decision stage. To each step $t \in T$, a DP algorithm must associate a set of decisions. In our case, decisions available for task t correspond to the machines $m \in M(S(t))$ to which t can be assigned. Additionally, the decision to assign task t to no machine must be available, and we denote it by e_t. At step $t \in T$, taking decision $m \in M(S(t))$ boils down to assigning value 1 to variable x_{tm}, and if decision e_t is taken, no assignment is done, as x_{tm} is set to 0 by default. Assigning value 1 to decision variable x_{tm} in partial solution x will be denoted by $x[x_{tm} \leftarrow 1]$. Let $Q_t \subseteq \{0,1\}^n$ be a subset of solutions built by taking decisions in periods 1 to t included, where only variables with indices (t', m'), for $t' \in \{1, ..., t\}$ and $m \in M(S(t'))$ have been fixed according to decisions taken at these steps, while other variables keep value 0.

In general, for $t \in T$, consider $x \in Q_t$ and for $\Delta_t^{\tilde{t}} := \prod_{t'=t+1}^{\tilde{t}} M(S(t')) \cup \{e_{t'}\}$, let $\delta \in \Delta_t^{\tilde{t}}$ denote a sequence of decisions from step $t+1$ to step $\tilde{t}$. We denote by $\tau(x, \delta)$ the solution obtained by taking decisions prescribed by decision sequence δ. Finally $Comp(x)$ and $Ext(x)$ denote respectively the set of completions and the set of extensions of partial solution x:

$$Comp(x) := \left\{\delta \in \Delta_t^{\tilde{t}} \mid \tau(x,\delta) \in X\right\}$$
$$Ext(x) := \{\tilde{x} \in X \mid \exists \delta \in Comp(x), \tilde{x} = \tau(x,\delta)\}$$

To the final step $\tilde{t}$ we associate the multiobjective function $f = (f_1, ..., f_p)$, and to each step $t \in \{1, ..., \tilde{t}-1\}$ we associate objective function $(f, \bar{w})$, where f is the multiobjective function in the original formulation of REF, and for any $x \in X$, $\bar{w}(x) = b - w(x) \in \mathbb{R}^{|S|}$, is the function evaluating the residual capacity associated with solution x, where with $b = (b_s | s \in S)$ is the bound vector of subsystem capacities.

Observation 4.1. *For any $t \leqslant \tilde{t}$, $x, x' \in Q_t$, any $\delta \in Comp(x) \cap Comp(x')$,*

$$f_j(x) \geqslant f_j(x') \quad \Rightarrow \quad f_j(\tau(x,\delta)) \geqslant f_j(\tau(x',\delta)) \quad \forall j \in \{1, ..., p\}$$

and

$$\bar{w}_s(x) \geqslant \bar{w}_s(x') \quad \Rightarrow \quad \bar{w}_s(\tau(x,\delta)) \geqslant \bar{w}_s(\tau(x',\delta)) \quad \forall s \in S$$

Since several solutions can have the same value vector, a solution is not defined by its value vector. Iin a DP decision sequence however, the only information relevant to the sequential construction of a non-dominated set is the information defining which assignments it can still receive, and which value it can attain, and these will be inferred from additional criteria, as in Bazgan et al. (2009). Thus two solutions with the same evaluation by the objective functions associated with the step at which they are generated are interchangeable from the point of view of DP. For this reason, $\tilde{\mathcal{E}}$ denotes a function that associates, to a each point of a non-dominated set, a single pre-image of this point.

Example 4.2 illustrates the general form and execution of a dynamic programming decision process.

Example 4.2. *The following example illustrates a dynamic programming decision process, through the state-space graphs. At decision stages t_1 and t_2, states Q_1 and Q_2 are generated by making feasible decisions from previously generated partial solutions, and then keeping only the non-dominated newly completed partial solutions.*

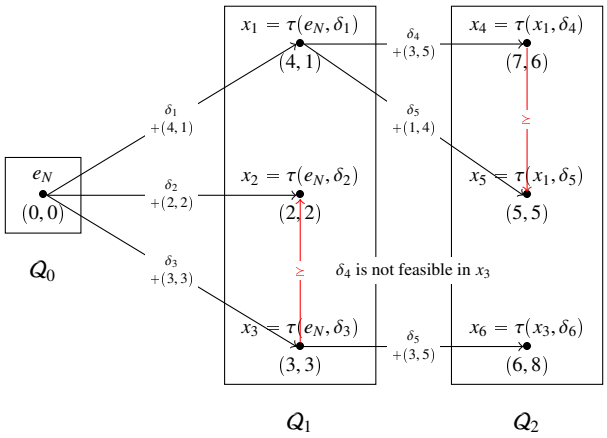


Figure 4.3: Visualization of state-space graph of a dynamic programming decision process. $T = \{t_1, t_2\}$. At stage t_1, x_3 dominates x_2, meaning that all the extensions of x_3 will dominate extensions of x_2, so we avoid considering extensions of x_2. We assume that decision δ_4 cannot be made from x_3, so that Q_2 contains no element $\tau(x_3, \delta_4)$. At stage 2, x_5 is dominated by x_4. Since t_2 is, in this case, the final stage, $\{x_4, x_6\}$ is the set of efficient solutions.

Algorithm 16 describes the resolution of REF using dynamic programming. First, we initialize Q_0 with the zero partial solution $\{e_N\}$. Then for each task $t \in T$, we initialize Q_t with value Q_{t-1}, since in REF, the decision of not assigning task t is always feasible. Then for each partial solution $x \in Q_{t-1}$, we generate all assignments of tasks t which extend x and do not violate the resource constraint, i.e. have non-negative residual capacity $\bar{w}$. Q_t must then be filtered by dominance. If t is not the last step, we need to preserve state-elements which are not dominated for residual capacities. Only if $t = \tilde{t}$ do we eliminate elements dominated according to f.

Algorithm 16: Dynamic Programming Algorithm for REF

input : $Q_0 \leftarrow \{e_N\}$
output: $\tilde{\mathcal{E}}(X, f)$

1 **for** $t \in \{1, ..., \tilde{t}\}$ **do**
2 $Q_t \leftarrow Q_{t-1}$
3 **for** $x \in Q_{t-1}$ **do**
4 **for** $m \in M(S(t))$ **do**
5 $\tilde{x} \leftarrow \tilde{x}[x_{tm} \leftarrow 1]$
6 /*Recall that for $m' \in M(S(t))\backslash\{m\}$ we already have $x_{tm} = 0$ */
7 **if** $\bar{w}(\tilde{x}) \geqq 0$ **then**
8 $Q_t \leftarrow Q_t \cup \{\tilde{x}\}$
9 **if** $t < \tilde{t}$ **then**
10 $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, (f, \bar{w}))$
11 **else**
12 $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, f)$
13 **return** $Q_{\tilde{t}}$

Note that because 0 is the default value of any variable, we do not have to explicitly generate partial solutions obtained by taking the neutral decision, as long as we begin each step $t \in T$ by setting $Q_t \leftarrow Q_{t-1}$. Next we show the correctness of this algorithm, i.e. that its produces the desired result:

Proposition 4.2. *Algorithm 16 returns the set $Q_{\tilde{t}}$ such that*

$$Q_{\tilde{t}} = \tilde{\mathcal{E}}(X, f)$$

Proof. $\supseteq$: Let $x \in \tilde{\mathcal{E}}(X, f)$, assume $x \notin Q_{\tilde{t}}$. Then there is some t' such that t' is the last step of the decision process at which some solution x' such that $x \in Ext(x')$ also belongs to $Q_{t'}$. If x' does not belong to $Q_{t'+1}$, then there is some $x'' \in Q_{t'}$ such that $(f, \bar{w})(x'') \geq (f, \bar{w})(x')$. Consider $\delta \in Comp(x')$ such that $\tau(x', \delta) = x$. Because $\bar{w}(x'') \geq \bar{w}(x')$, we also have $\delta \in Comp(x')$. Then because $f(x'') \geq f(x')$, by Observation C.4, we also have, for $x^* = \tau(x'', \delta)$, that $f(x^*) \geq f(x)$. And thus, $x \notin \tilde{\mathcal{E}}(X, f)$, contradicting our assumption.
$\subseteq$: Let $x \in Q_{\tilde{t}}$, and assume $x \notin \tilde{\mathcal{E}}(X, f)$. This means that there exists $x' \in X$ such that $f(x') \geq f(x)$. We can assume that $x' \in \tilde{\mathcal{E}}(X, f)$, otherwise we replace it with the maximal element of a $\geq$ chain it belongs to. Because $\tilde{\mathcal{E}}(X, f) \subseteq Q_{\tilde{t}}$, $x' \in Q_{\tilde{t}}$, and thus $x \notin Q_{\tilde{t}}$, being eliminated at line 12, contradicting our assumption. □

Algorithm 16 will be used as a subprocedure within many algorithms that are presented in this work. Thus, we use the following notation to describe an instance of REF and an application

p	$\lvert M(s)\rvert$	$\lvert T_C\rvert$	$\lvert S\rvert$	$\lvert N\rvert$	$\lvert ND\rvert$		$T.\ e-const$ *NoDec*		$T.\ DP$ *Basic*	
2	4	2	2	48	17.950	[8.056]	0.640	[0.541]	0.434	[0.273]
2	5	2	2	70	23.150	[7.107]	0.976	[0.536]	3.316	[2.262]
2	6	2	2	96	39.050	[10.737]	2.519	[0.788]	12.896	[8.228]
2	4	3	2	56	19.100	[6.718]	0.662	[0.352]	1.390	[0.521]
2	4	4	2	64	24.800	[5.877]	0.846	[0.340]	2.420	[0.792]
2	4	5	2	72	29.350	[10.143]	1.333	[0.818]	5.299	[2.085]
2	4	2	3	72	25.500	[11.826]	1.314	[0.780]	20.940	[11.469]
2	3	1	4	48	11	[5.560]	0.317	[0.275]	5.010	[1.954]
2	3	2	4	60	25	[11.152]	1.006	[0.657]	35.278	[14.205]
3	5	2	2	70	173.800	[57.036]	19.534	[5.435]	8.633	[3.488]
3	5	3	2	80	467.800	[278.135]	60.296	[31.120]	28.138	[17.006]
3	4	2	3	72	203	[113.837]	27.269	[16.142]	136.061	[115.563]

Table 4.1: Computing time (in seconds) using e-constraint and using Direct DP, for various instance parameter values and $p = 2, 3, 4$. Average values for 10 trials.

of Algorithm 16. We denote an instance by the tuple (Q_0, T_C, T_L, M, S), and we write $B \leftarrow DP(Q_0, T_C, T_L, M, S)$ when we obtain a pre-image $\tilde{\mathcal{E}}(X, f)$ of $\mathcal{N}(f(X))$ by solving the instance using Algorithm 16.

4.1.3.1 Experimental results

In all our experiments on problem REF, instances are generated as follows. We ensure that if $|S(t)| > 1$, then $S(t) = S$ for any $t \in T$. In other words, if a task is complex, it can be performed on *all* sites. This is a way to both avoid considering cases where intermediary decomposition into several coupled subproblems would be possible, and to make sure that our performance measurements pertain to hard instances. For each $(t, m) \in N$ and $j \in \{1, ..., p\}$, g^j_{tm} and w_{tm} are both drawn uniformly randomly from interval $[0, 1000]$. Finally, we compute b_s the bound of the capacity of site $s \in S$ so as to adapt the notion of "hard" knapsack constraints, setting

$$b_s = \frac{1}{2} \frac{\sum_{t \in T(s), m \in M(S(t))} w_{tm}}{|T(s)|}$$

In the experiment reported in Table 4.1, we solve instances of REF for various values of p and instance parameters $|M(s)|$, $|T_L(s)|$, $|T_C|$, and $|S|$. Unless otherwise stated, we will assume that $|M(s)|$ has the same value for all $s \in S$, and that $|M(s)| = |T_L(s)|$, i.e. that the subsystem assignment matrix is square. $T.e-const$ denotes the computing time to solve the instance using either biobjective e-constraint method for $p = 2$ or the generic algorithm by Tamby and Vanderpooten (2021) for $p \geqslant 3$. Finally, *T.DP Basic* denotes computing time for the dynamic programming algorithm.

Experiments reveal the execution of this method to yield contrasted results. It performs well relatively to $e-const$ when p increases, but way more poorly as $|T_C|$ or $|S|$ increase. The main reason for the increased computational cost, in the latter case, is the $p + |S|$ dimensionality of DP

states, which results in the procedure generating a high number of mostly incomparable states until the very last stage of the process, where dimensionality is reduced to p. Several improvements can be made upon it to gain some speed, as will be demonstrated later by an experimental protocol comparing this basic approach to resolution methods that benefit from some improvements using decomposition. But as these tricks use upper and lower bound sets, we need first to define those for REF, and describe algorithms to obtain them.

4.2 Upper and lower bound sets for REF

Weak upper and lower bound notions developped in Chapter 3 can be straightforwardly applied to REF. Furthermore, these bound sets can be computed using dynamic programming algorithms.

4.2.1 Weak upper bound sets and a strong singleton upper bound

4.2.1.1 Copy-split relaxation

We recall that in the *copy-split* relaxation, each subsytem problem inherits a copy of each coupling constraints, which is restricted to variables associated with complex tasks assignments than can be made in it. In the notation introduced in Section C.3.3.1, we assume that coupling constraints imposed by coupling $k \in K$ can be reformulated as $g_k(x^k) \leqslant b_k$ for some real valued function g. We further assume that for $j \in \{1, ..., p\}$, f_j is additively separable along S and that so is g_k for any $k \in K$. Recall that for any $k \in K$, $\gamma(k) = \{s \in S \mid s \cap k \neq \varnothing\}$. Recall that assuming that coupling constraint k is relaxable for each $s \in |\gamma(k)|$, we denote by $g_k'^s(x_k^s) \leqslant b_k$ the relaxation of $g_k(x_k) \leqslant b_k$ associated with $s \in |\gamma(k)|$. Then the copy split relaxation can be expressed as:

$$\begin{array}{lll} \max & f(x) & \\ s.t. & g_k'^s(x_k^s) \leqslant b_k & \forall\, k \in K, s \in \gamma(k) \\ & x^s \in X^s & \forall\, s \in S \end{array}$$

In REF, $K = \{(t, m) \mid t \in T_C, m \in M(S(t))\}$ and $g_k^s(x_k^s) = \sum_{m \in M(S(t))} x_{tm}$ for $t \in T_C$. For any $t \in T$, $b_t = 1$. Then the copy-split relaxation of REF can be formulated as follows:

$$\begin{array}{lll} \max & \displaystyle\sum_{t \in T} \sum_{m \in M(S(t))} g_{tm}^j x_{tm} & \forall\, j \in \{1, ..., p\} \\ s.t. & \displaystyle\sum_{m \in M(S(t))} x_{tm} \leqslant 1 & \forall\, t \in T,\ s \in S(t) \\ & \displaystyle\sum_{t \in T(s)} \sum_{m \in M(S(t))} w_{tm} x_{tm} \leqslant b_s & \forall\, s \in S \\ & x_{tm} \in \{0, 1\} & \forall\, t \in T,\ m \in M(S(t)) \end{array}$$

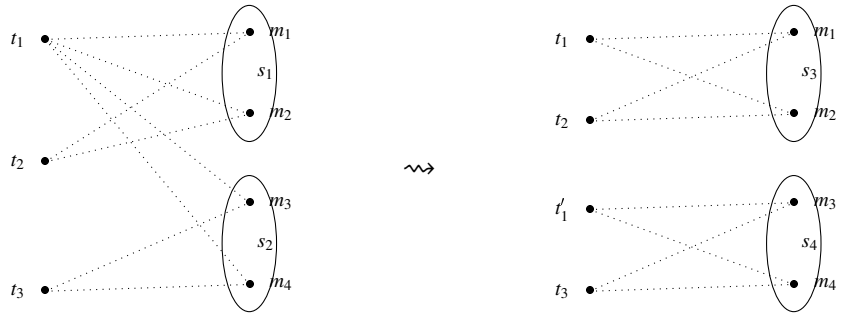


Figure 4.4: Visualization of the copy-split relaxation as applied to REF: assignments of complex task t_1 are duplicated and made simple tasks assignments in each subsystem

Because the copy-split relaxation is separable, this is equivalent to solving $|S|$ different instances of REF for each $s \in S$, having copied each complex task so that a copy of it appears as a local task in each subproblem, as is illustrated by Figure C.11. We are left with $|S|$ subproblems of the form:

$$\begin{aligned}
\max \quad & \sum_{t \in T(s)} \sum_{m \in M(S(t))} g_{tm}^{j} x_{tm} && \forall\, j \in \{1, ..., p\} \\
s.t. \quad & \sum_{m \in M(s)} x_{tm} \leqslant 1 && \forall\, t \in T(s) \\
& \sum_{t \in T(s)} \sum_{m \in M(S(t))} w_{tm} x_{tm} \leqslant b_s \\
& x_{tm} \in \{0, 1\} && \forall\, t \in T(s),\ m \in M(S(t))
\end{aligned}$$

As this subproblem is itself an instance of REF, it can be solved using Algorithm 16. Then, we compute *copy-split* relaxation using Algorithm 17.

Algorithm 17: Dynamic Programming Algorithm for solving the copy-split relaxation of REF with decomposition.

input : $(\{e_N\}, T_C, T_L, M, S)$

$SubRes \leftarrow (\varnothing_1, ..., \varnothing_{|S|})$

for $s \in S$ **do**

  $T'_L(s) \leftarrow T(s)$

  $SubRes_s \leftarrow DP(\{e_N\}, \varnothing, T'_L(s), M(s), \{s\})$

return $Pool(SubRes)$

4.2.1.2 Relaxation of complex task assignment variables from non-coupling constraints

As shown in Section C.3.3.2, a less obvious way to relax the problem is to relax, from local constraints, decision variables which appear in coupling constraints, a relaxation which we have

called the *LR* relaxation. We recall that for $\bar{K}$ the set of non-coupled variables, $g^s(x^s) \leqslant b^s$ is relaxed into $g'^s(x^s_{\bar{K}}) \leqslant b^s$. Applying this relaxation to all $s \in S$, this variant is formulated as follows,

$$\begin{array}{lll} \max & f(x) & \\ s.t. & x^k \in X^k & \forall\, k \in K \\ & g'^s(x^s_{\bar{K}}) \leqslant b^s & \forall\, s \in S \end{array}$$

Thus, for $s \in S$, x^s_K is not constrained in subproblem associated with site s. In REF, $K = \{(t,m) \mid t \in T_C, m \in M(S(t))\}$, and $\bar{K} = \{(t,m) \mid t \in T_L, m \in M(S(t))\}$, Applied to problem REF, this relaxation scheme yields the following problem:

$$\begin{array}{lll} \max & \displaystyle\sum_{t\in T}\sum_{m\in M(S(t))} g^j_{tm}x_{tm} & \forall\, j \in \{1,...,p\} \\ s.t. & \displaystyle\sum_{m\in M(S(t))} x_{tm} \leqslant 1 & \forall\, t \in T_C \\ & \displaystyle\sum_{m\in M(S(t))} x_{tm} \leqslant 1 & \forall\, s \in S,\ t \in T_L(s) \\ & \displaystyle\sum_{t\in T_L(s)}\sum_{m\in M(S(t))} w_{tm}x_{tm} \leqslant b_s & \forall\, s \in S \\ & x_{tm} \in \{0,1\} & \forall\, t \in T,\ m \in M(S(t)) \end{array}$$

This problem is separated in $|S| + 1$ subproblems, the first $|S|$ of which involve only tasks local to system $s \in S$. Each of these tasks must be assigned to at most one machine of site s under the budged constraints of s :

$$\begin{array}{lll} \max & \displaystyle\sum_{t\in T_L(s)}\sum_{m\in M(s)} g^j_{tm}x_{tm} & \forall\, j \in \{1,...,p\} \\ s.t. & \displaystyle\sum_{m\in M(s)} x_{tm} \leqslant 1 & \forall\, t \in T_L(s) \\ & \displaystyle\sum_{t\in T_L(s)}\sum_{m\in M(S(t))} w_{tm}x_{tm} \leqslant b_s & \\ & x_{tm} \in \{0,1\} & \forall\, t \in T_L(s),\ m \in M(S(t)) \end{array}$$

Each of these problems can be solved using DP, computing $DP(\{e_N\}, \varnothing, T_L(s), M(s), \{s\})$, for $s \in S$. Then, there remains the now-independent problem of assigning each complex task to at most one machine, under no budget constraint:

$$
\begin{aligned}
\max \quad & \sum_{t \in T_C} \sum_{m \in M(S(t))} g_{tm}^{j} x_{tm} && \forall\, j \in \{1, ..., p\} \\
s.t. \quad & \sum_{m \in M(S(t))} x_{tm} \leqslant 1 && \forall\, t \in T_C \\
& x_{tm} \in \{0, 1\} && \forall\, t \in T_C,\ m \in M(S(t))
\end{aligned}
$$

This problem, the *complex task relaxation problem*, can also be solved using a DP algorithm, albeit a much simpler one that the one used for solving subproblems, because there is no need to check for the feasibility of partial solutions with regards to budget constraints. This simplified procedure is presented as Algorithm 18, the application of which is denoted by $DP(\{e_N\}, T_C, \varnothing, M, S)$.

Algorithm 18: Dynamic Programming Algorithm for the *complex tasks* assignemnts in the *LR* relaxation of REF

input : $(\{e_{T_C}\}, T_C, \varnothing, M, S)$

$Q_0 = \{e_{T_C}\}$

/*Assume $|T_C| = \tilde{t}$ */

for $t \in \{1, ..., \tilde{t}\}$ **do**

  for $x \in Q_{t-1}$ **do**

    for $m \in M(S(t))$ **do**

      $\tilde{x} \leftarrow \tilde{x}[x_{tm} \leftarrow 1]$

      $Q_t \leftarrow Q_t \cup \{\tilde{x}\}$

    $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, f)$

return $Q_{\tilde{t}}$

Thus, Algorithm 19 solves the LR relaxation by pooling results of the locally restricted subproblems and the complex tasks problem, using any pooling algorithm, here denoted generically by *Pool*.

Algorithm 19: Dynamic Programming Algorithm for solving the *LR* relaxation of REF

input : $(\{e_N\}, T_C, T_L, M, S)$

$Complex \leftarrow DP(\{e_N\}, T_C, \varnothing, M, S)$ /*Compute non-dominated complex tasks assignments */

$Local \leftarrow \varnothing$

$SubRes \leftarrow (\varnothing_1, ..., \varnothing_{|S|})$

for $s \in S$ **do**

  $SubRes_s \leftarrow DP(\{e_N\}, \varnothing, T_L(s), M(s), \{s\})$

$Local = Pool(SubRes)$

return $Pool(Local, Complex)$

4.2.1.3 Singleton strong upper bound

Improvements on the basic resolution method for REF can be made by eliminating partial solutions when a *strong* upper bound on the value of their extensions is found to be dominated by an incumbent point. Because these must be computed quite often, we look for a greedy algorithm to obtain them.

In the case of the single objective knapsack problem, it is well known (see e.g. Kellerer et al. (2004)) that the linear relaxation upper bound can be obtained using the following greedy algorithm. Objects $t \in \{1, ..., \tilde{t}\}$ are ranked in decreasing order of ratio $\frac{g_t}{w_t}$. Objects are added in order until residual capacity becomes negative, and the resulting sum of profits is an upper bound on the value of the optimal knapsack. This can be improved further using Martello and Toth (1990)'s concept of upper bound. This concept of singleton upper bound can be applied to any partial solution where objects $\{1, ..., t\}$ have been previously decided upon, for $t < \tilde{t}$. Objects upon which no decision has yet been taken are then added, in an order determined by the previously defined ratio. Finally, point $(u_1^G, ..., u_1^G)$ obtained by computing the objective-wise greedy upper bounds is a *strong* upper bound on all non-dominated knapsacks, or extensions of a partial knapsack.

However none of these bound concepts straightforwardly applies to our case, even when assuming that $|S| = 1$, as is evident from the following counter example.

Example 4.3. *Let $T = \{1, 2\}$, $S = \{s\}$, $M(s) = \{a, b\}$, assume that $b_s = 6$ and consider the following matrix, where an entry reads as g_{tm}/w_{tm}, for $t \in T$ and $m \in M(s)$.*

	1	*2*
a	*20 / 4*	*2/4*
b	*24/ 6*	*1/4*

The following sequence of pairs describes the ranking of assignments in decreasing order of ratio $\frac{g_{tm}}{w_{tm}}$: $(1, a), (1, b), (2, a), (2, b)$. The greedy algorithm takes $(1, a)$, after which the knapsack has value of 20 and a residual capacity of 2. $(1, b)$ cannot be added to the knapsack because task 1 has already been assigned to machine b. Then, all remaining assignments would violate the knapsack contraints, so the knapsack is full. The candidate upper bound value of 22 is obtained by further adding the value of $(2, a)$, the split assignment. However, the optimal solution to this instance of the problem is to select $(1, b)$, for a profit of 24, which is higher than 22. Thus, the greedy algorithm failed to yield an upper bound solution.

We propose a greedy heuristic which gets around the issue of the assignment constraint. First, we copy complex tasks, so that we can decouple the problem as in the *copy-split* relaxation. This yields T' such that, for each $t \in T_C$, for each $s \in S(t)$, and $m \in M(s)$, we have $t' \in T'_L(s)$ with $w_{t'm} = w_{tm}$ and $g^j_{t'm} = g^j_{tm}$ for each $j \in \{1, ..., p\}$. Then we replace T' with a set of artificial tasks

	1	2
g_t^{j*}/w_t^*	24 / 4	2/4

Figure 4.5: Profits and weights associated with T^* when applied to Example 4.3

T^*, such that for all $t \in T'$, $m \in M(t')$,

$$g_t^{j*} = \max_{m' \in M(t)} g_{tm'}^j$$
$$w_t^* = \min_{m' \in M(t)} w_{tm'}$$

For each $j \in \{1, ..., p\}$, we compute the j-th component of the objective-wise optimum as follows. For each $s \in S$, we rank $T^*(s)$ in decreasing order of $\frac{g_t^{j*}}{w_t^*}$, and we compute a candidate objective-wise upper bound by making assignments in order, until the assignment which exceeds capacity of site s has been made. We denote by u_j^* the objective value of this greedy solution on component j.

Proposition 4.3. *$(u_1^*, ..., u_p^*)$ computed by the previously described procedure is a singleton strong upper bound to the set of extensions of a partial solution.*

Proof. Consider, for any instance of REF, its copy-split relaxation, with feasible set X' and objective function f'. Then consider the variant of the copy-split relaxation where for any $(t, m) \in N$, g_{tm}^j is replaced by g_t^{j*} and w_{tm} by w_t^*. Since for any $(t, m) \in N$, we have $w_t^* \leqslant w_{tm}$ and $g_t^{j*} \geqslant g_{tm}^j$, this variant with feasible set X'' and objective function f'' is a relaxation of the instance with feasible set X'. $\max_{x \in X'}(f_1'(x), ..., f_j'(x))$ is itself a relaxation of the instance of REF, and by transitivity $\max_{x \in X''}(f_1''(x), ..., f_j''(x))$ is a relaxation of X.

Now, let us show that $(u_1^*, ..., u_p^*)$ is a singleton strong upper bound over the set $\mathcal{N}(f''(X''))$. Indeed, for $s \in S$, the subproblem associated with s of the problem with feasible set X'' is equivalent to an instance of the knapsack problem, since the assignments of a task $t \in T$ now all yield the same profit and have the weight value for any accessible machines in $M(t)$. For each $j \in \{1, ..., p\}$, component u_j^* of greedy assignments according to decreasing order of $\frac{g_t^{j*}}{w_t^*}$ (where the limit item is taken in the knapsack) is an upper bound to the linear relaxation of the same knapsack problem. Therefore, $u_j^* \geqslant \max_{x \in X'} f_j''(x)$, and by transitivity applied twice, $u_j^* \geqslant \max_{x \in X} f_j(x)$ for all $j \in \{1, ..., p\}$. □

4.2.2 Strong and weak lower bound sets

4.2.2.1 A strong lower bound set

We consider the feasible set X_L of the restriction of the original problem to local task variables, i.e. $X_L = \{x \in X \mid x_{tm} = 0 \ \forall t \in T_C, m \in M(S(t))\}$ As a particular case of the notions we

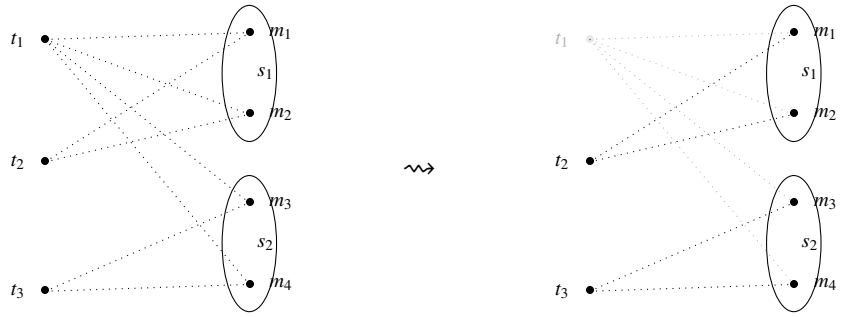


Figure 4.6: Visualization of the local restriction as applied to REF: assignments of complex task t_1 are removed from all subsystems

studied in Chapter 3, it is clear that $\mathcal{N}(f(X_L))$ is a weak lower bound on $\mathcal{N}(f(X))$. Furthermore, the computation of $\tilde{\mathcal{E}}(X_L,(f,\bar{w}))$ is separable, since $X_L = \Pi_{s\in S} X_{L(s)}$ for $X_{L(s)} = \{x^s \in X^s \mid x^s_{tm} = 0\ \forall t \in T_C, m \in M(s)\}$. When computing the efficient subset of X_L for f, we eliminate all solutions dominated for the p original criteria, even if these solutions are non-dominated when residual capacity is taken into account. If, to the contrary, we keep the latter solutions, but still *evaluate* them according to the p original criteria, we obtain set $f(\tilde{\mathcal{E}}(X_L,(f,\bar{w}))) \subseteq \mathbb{R}^p$, which lies in the same objective space as $\mathcal{N}(f(X))$ and has the following property:

Proposition 4.4. *$f(\tilde{\mathcal{E}}(X_L,(f,\bar{w})))$ is a strong lower bound set to $\mathcal{N}(f(X))$.*

Proof. Assuming $T_L = \{1,...,l\}$, $\tilde{\mathcal{E}}(X_L,(f,\bar{w}))$ corresponds to Q_l in Algorithm 16. This means that for any $x \in \tilde{\mathcal{E}}(X,f)$, there exists $x' \in Q_l$ such that $x \in Ext(x')$. Therefore, we have $f(x) \geq f(x')$. Thus for any $f(x) \in \mathcal{N}(f(X))$, there exists $f(x') \in f(\tilde{\mathcal{E}}(X_L,(f,\bar{w}))$ such that $f(x) \geq f(x')$, establishing the result. □

Further on, we will call this strong lower bound *Strong LocRes*, as opposed to the weak variant of the same restriction, which has the same set of feasible solutions, but is solved only for the main p criteria. Let us recall that a strong lower bound set is not necessarily better than a weak lower bound set when used as an incumbent set. Rather, it is a guarantee that all non-dominated points lie in the union of the upper orthants associated with its elements, rather than in the incomparable regions *around* it. In practice, because $\mathcal{E}(X_L)$ contains close-to-empty-solutions which maximize residual capacity, its image $\mathcal{N}(f(X_L))$ is not a very useful strong lower bound set. However, using *singleton upper bounds*, computed by taking into account all remaining complex tasks, i.e. all $t \in T_C$, we can improve on it significantly. Namely, we can remove from $\tilde{\mathcal{E}}(X_L,(f,\bar{w}))$ any solution, such that a strong upper bound on its set of extensions is dominated by another solution from the same set. Formally:

Proposition 4.5. *Let $x \in \mathcal{E}(X_L,(f,\bar{w}))$, u^x a singleton strong upper bound on $Ext(x)$, and $\succeq_u := \{(x',x) \in X^2 \mid f(x') \geq u^x\}$. Then $f(\mathcal{E}(\tilde{\mathcal{E}}(X_L,(f,\bar{w})),\succeq_u))$ is a strong lower bound set on $\mathcal{N}(f(X))$. In other words, filtering $\mathcal{E}(X_L,(f,\bar{w}))$ by $\succeq_u$ yields a strong lower bound set to $\mathcal{N}(f(X))$.*

Proof. By Proposition 4.4, for each $x \in \tilde{\mathcal{E}}(X,f)$, there exists $x' \in \tilde{\mathcal{E}}(X_L,(f,\bar{w}))$ such that $x \in$

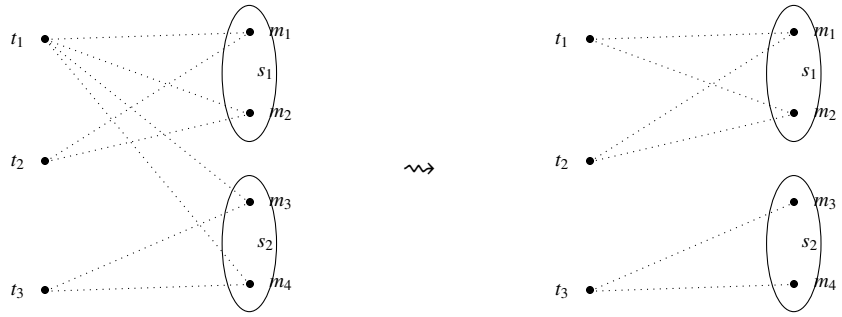


Figure 4.7: Visualization of the restrict-split restriction as applied to REF: assignments of complex task t_1 are restricted to site s_1. In this case, we have $T^*_{s_1} = \{t_1\}$ and $T^*_{s_2} = \varnothing$

.

$Ext(x')$. For $x \in \mathcal{E}(X, f)$, let $Rest(x, L) := \{x' \in X_L \mid x \in Ext(x')\}$, and $Rest(\mathcal{E}(X, f), L) = \bigcup_{x \in \mathcal{E}(X,f)} Rest(x, L)$. As long as filtering by $\succeq_u$ removes no element from $Rest(\mathcal{E}(X, f), L)$, it yields a strong lower bound set. For any $x' \in \mathcal{E}(X_L, (f, \bar{w}))$, $u^{x'}$ is a strong upper bound on $Ext(x')$, meaning that for any $x \in Ext(x'), u^{x'} \succeq x$. Thus for any $x', x'' \in \mathcal{E}(X_L, (f, \bar{w}))$ if $f(x'') \succeq u^{x'}$, then there cannot be any $x \in \mathcal{E}(X, f)$ such that $x \in Ext(x')$, and thus $x' \notin Rest(\mathcal{E}(X, f), L)$. □

4.2.2.2 Weak lower bound obtained from solving the *Restrict-split* variant

The adaptation of the *restrict-split* variant is fairly straightforward. For each $t \in T_C$, we choose one $s \in S$, and thus for each $s \in S$ we obtain a subset of $T^*_s \subseteq T_C$ of complex tasks which become local tasks for s and are thus added to the set of tasks $T_L(s)$, as seen line 4 in Algorithm 20 and illustrated by Figure 4.7. Profits and weights of assignments which are kept from the original instance by this operation remain unchanged.

Algorithm 20 describes the computation of the restrict-split variant of REF using dynamic programming. Once we have computed the heuristic assignment of complex tasks to subsystems, we modify each subsystem instance, solve them independently, and pool the resulting solution sets together.

Algorithm 20: Dynamic Programming Algorithm for solving the copy-split relaxation of REF with decomposition.

input : $(\{e_N\}, T_C, T_L, M, S)$
output: $\mathcal{N}(f(X'))$
1 $SubRes \leftarrow (\varnothing_1, ..., \varnothing_{|S|})$
2 Let $(T^*_s \mid s \in S)$ be the list of subsets of complex tasks restriced to each $s \in S$
3 **for** $s \in S$ **do**
4 $\quad SubRes_s \leftarrow DP(\{e_N\}, \varnothing, T_L(s) \cup T^*_s, M(s), \{s\})$
5 **return** $Pool(SubRes)$

4.2.2.3 Weak lower bound sets obtained from greedily completing partial solutions

As a weak LB, the image of *weak LocRes*, i.e. $f(\tilde{\mathcal{E}}(X_L, f))$ (or alternatively $\mathcal{N}(f(X_L))$) would perform poorly, but it can easily be improved. Indeed, it may contain solutions which still have enough residual capacity to receive assignments of complex tasks. Even more such assignments could be made on solutions of *some subset* of *strong LocRes*, i.e. some subset of $\tilde{\mathcal{E}}(X_L, (f, \bar{w}))$, but doing all possible non-dominated assignments on the whole *strong LocRes* would amount to solving the original problem, and our goal here is only to obtain a weak LB. We need to strike the right balance between the richness of the set from which we complete solutions, the speed of computing extensions, and the tightness of the obtained weak LB.

Let $\kappa(B)$ denote the set of solutions obtained by completing partial solutions from some set B with further feasible assignments. In order to obtain an incumbent set quickly, κ should be computable by a greedy algorithm. We use a ranking of complex tasks assignments variables to complete, when possible, solutions obtained after combining solutions of the local problems. Following Bazgan et al. (2009), we sort assignments by decreasing order of $\frac{g^j_{tm}}{w_{tm}}$, for each objective $j \in \{1, ..., p\}$. We then globally sort the assignments in increasing order of the sum of their ranks in objective-wise orders, as described by Algorithm 21.

For each solution $x \in B$, we make feasible assignments of tasks to machine according to the previously defined ranking. Because a task can be assigned only once and within site budget constraint, when a task has been assigned, we can clear the ranked list of its other assignments. We remove all assignments to other machines of the task having just been assigned, and all assignments to machines on a site that would have been saturated at that step.

It appears, from preliminary experiments, that $\kappa(\tilde{\mathcal{E}}(X_L, (f, \bar{w})))$ yields a far richer incumbent set than $\kappa(\tilde{\mathcal{E}}(X_L, f))$, because the solutions it contains admit more completions from additional variable assignments, since they belong to this set because of their low budgets consumption. However, computing $\kappa(\tilde{\mathcal{E}}(X_L, (f, \bar{w})))$ without a preliminary filtering proves very expensive, while many quasi-empty solutions cannot get good enough completions. We can modulate the richness of the set on which to apply greedy completions in three ways.

First, remembering that $X_L = \prod_{s \in S} X_L$, we may compute, for each $s \in S$, $\tilde{\mathcal{E}}(X^s_L, f)$ rather than $\tilde{\mathcal{E}}(X^s_L, (f, \bar{w}))$. Second, for B^s the set of solutions computed for decomposable steps associated with $s \in S$, rather than computing $\mathcal{N}((f, \bar{w})(\prod_{s \in S} B^s))$, we may compute $\mathcal{N}(f(\prod_{s \in S} B^s))$. Note that even if we have computed $\tilde{\mathcal{E}}(X^s_L, f)$ as suproblem solutions, we can still compute $\tilde{\mathcal{E}}((\prod_{s \in S} B^s), (f, \bar{w}))$, which will be richer than $\tilde{\mathcal{E}}((\prod_{s \in S} B^s), f)$.

Third, we can require that the f-dominated solutions to be completed by the greedy algorithm still score better than some minimal f values. Indeed, many solutions from a sufficiently rich set, be it either $\mathcal{E}((\prod_{s \in S} \tilde{\mathcal{E}}(X^s_L, f)), (f, \bar{w}))$ or $\mathcal{E}((\prod_{s \in S} \tilde{\mathcal{E}}(X^s_L, (f, \bar{w})), (f, \bar{w}))$ must still have too low values on f for their completions to yield non-dominated solutions. We can filter them out using an approach akin to $\varepsilon - dominance$. We will say that x $\alpha-$dominates x' if for all $j \in \{1, ..., p\}$, $f_j(x) \geqslant f_j(x') + \alpha * M$, where M could be the maximum possible profit value, or could be $f_j(x')$.

Algorithm 21: Greedy wLB from completions of solutions to LocRes

input : $B \subseteq X_L, Asg$

/*Asg denotes the list of ranked assignment variable indices */

output: $\kappa(B)$

$OutSet = \{\}$

for $x \in B$ **do**

  while $\tilde{x} \in X$ & $Asg \neq \varnothing$ **do**

    for $(t,m) \in Asg$ **do**

      $\tilde{x} \leftarrow \tilde{x}[\tilde{x}_{tm} \leftarrow 1]$

    $Asg \leftarrow Asg \backslash \{(t',m) \in Asg \mid t' = t\}$ /*Remove all assignments of task t to other machines to preserve feasibility */

    for $s \in S$ **do**

      if $\bar{x}_s < 0$ **then**

        $Asg \leftarrow Asg \backslash \{(t,m) \in Asg \mid m \in M(s)\}$ /*If capacity of site s is exceeded, remove all assignments of remaining tasks to machines of site s. */

  $OutSet \leftarrow OutSet \cup \{\tilde{x}\}$

return $OutSet$

We denote this filtering operation by $\tilde{\mathcal{E}}(B^*, \alpha(f))$. Finally one has to choose an α. A relevant value is one such that the obtained incumbent, $\kappa(\mathcal{E}(B^*, \alpha(f)))$, is nearly as rich as $\kappa(\tilde{\mathcal{E}}(B^*, (f, \bar{w}))$, but is obtained in a sufficiently short time.

4.2.3 Experimental Results

The following experiments compare the time necessary to compute two weak lower bounds: the solution of the *restrict-split* variant of REF, and the greedy lower bounds described in the previous section. We will also assess the quality of these bounds as approximations of the original REF instance. $|M(s)|$ will denote the number of machines on each site s, assuming that all $s, s' \in S$, $|M(s)| = |M(s')|$. Once again, the matrix of assignments of local tasks to machines of each site is square, with, for all $s \in S$, $|T_L(s)| = |M(s)|$. $|S|$ denotes the number of subsystems, hence of subproblems. $|N|$ is the number de decision variables in the instance.

In Table 4.2, $ApQ.(\varepsilon)$ denotes, as in Section C.3.4, the approximation quality of a set as measured by *a posteriori* ε-dominance. We did not perform experiments using the greedy completions of $\tilde{\mathcal{E}}(X_L, (f, \bar{w}))$, because computing time for this weak lower bound proved to be prohibitive for the instance sizes considered. Rather, we compared weak lower bounds obtained by greedily completings three types of base solutions sets.

First, *GrS LocRP* denotes $\mathcal{E}(\prod_{s\in S} \tilde{\mathcal{E}}(X_L^s, f), (f, \bar{w}))$, which is obtained by solving the subproblems for criteria f, but keeping the *combinations* of subsystem solutions which are non-dominated for $(f, \bar{w})$. Second, *GrS LocRP*α denotes $\mathcal{E}(\mathcal{E}(\prod_{s\in S} \tilde{\mathcal{E}}(X_L^s, f), (f, \bar{w})), \alpha(f))$, which is the same as the previous one, where the result of the pooling is further filtered according to α-dominance as described in Section 4.2.2.3. We use $\alpha = \frac{|T_C|}{|T_L|}$, and relation $x \succeq_\alpha x' \Leftrightarrow f(x) \geq f(x') + \alpha \times gMax$ where, for $j \in \{1, ..., p\}$, $gMax_j = 1000$ is the upper bound of the interval in which g_{tm}^j is randomly drawn, for any $(t, m) \in N$. Third, *GrWLocR* denotes the greedy completions of $\tilde{\mathcal{E}}(X_L, f)$, which is simply the *weak LocRes* lower bound.

p	$\lvert M(s)\rvert$	$\lvert T_C\rvert$	$\lvert S\rvert$	*GrS LocRP*		*GrS LocRP*α		*GrWLocR*	
				Time	$ApQ.(\varepsilon)$	*Time*	$ApQ.(\varepsilon)$	*Time*	$ApQ.(\varepsilon)$
2	7	1	2	0.312 [0.08]	0.037 [0.02]	0.227 [0.06]	0.044 [0.03]	0.192 [0.04]	0.047 [0.02]
2	8	1	2	0.708 [0.23]	0.033 [0.02]	0.559 [0.15]	0.037 [0.02]	0.466 [0.10]	0.039 [0.02]
2	9	1	2	1.531 [0.50]	0.030 [0.01]	1.162 [0.39]	0.035 [0.02]	1.008 [0.31]	0.038 [0.02]
2	6	1	2	0.102 [0.03]	0.043 [0.02]	0.075 [0.01]	0.047 [0.03]	0.067 [0.01]	0.050 [0.03]
2	6	1	3	0.956 [0.52]	0.031 [0.02]	0.341 [0.14]	0.033 [0.02]	0.233 [0.06]	0.036 [0.02]
2	6	1	4	6.762 [3.76]	0.020 [0.01]	1.063 [0.41]	0.025 [0.01]	0.588 [0.11]	0.026 [0.01]
2	7	2	2	0.434 [0.13]	0.062 [0.02]	0.334 [0.10]	0.069 [0.03]	0.251 [0.06]	0.078 [0.03]
2	7	3	2	0.483 [0.17]	0.099 [0.02]	0.427 [0.18]	0.104 [0.02]	0.282 [0.07]	0.115 [0.03]
3	4	2	2	0.164 [0.15]	0.127 [0.05]	0.133 [0.11]	0.129 [0.05]	0.077 [0.05]	0.143 [0.05]
4	4	2	2	1.159 [0.52]	0.135 [0.04]	0.939 [0.48]	0.137 [0.04]	0.545 [0.28]	0.148 [0.04]

Table 4.2: Computing time (second) and approximation quality of the weak lower bound using greedy completions of elements of the strong LocRes lower bound. Average values for 10 trials.

From Table 4.2, we can observe, first, the expected: the richer the set from which the completions are computed, the better the approximation quality ($ApQ.(\varepsilon)$, to be minimized), but the more expensive the computation becomes. Because the tradeoff between these two measurement

appears to be rather balanced, it is difficult to decide in advance which of these variants of greedy weak lower bound is the best.

p	$\lvert M(s)\rvert$	$\lvert T_C\rvert$	$\lvert S\rvert$	*RS LB*			
				Time DP		*ApQ.* (ε)	
2	7	1	2	0.194	[0.058]	0.019	[0.018]
2	8	1	2	0.524	[0.109]	0.017	[0.013]
2	9	1	2	1.120	[0.330]	0.006	[0.007]
2	6	1	2	0.063	[0.011]	0.022	[0.020]
2	6	1	3	0.166	[0.033]	0.019	[0.014]
2	6	1	4	0.396	[0.082]	0.017	[0.009]
2	7	2	2	0.286	[0.067]	0.024	[0.018]
2	7	3	2	0.412	[0.120]	0.035	[0.015]
3	4	2	2	0.016	[0.006]	0.095	[0.051]
4	4	2	2	0.030	[0.010]	0.105	[0.046]

Table 4.3: Computing time (second) and approximation quality of the weak lower bound obtained from solving the restrict-split restriction using DP. Average values for 10 trials.

Comparing results for the greedy lower bounds with results for the *restrict-split* variant in Table 4.3, it appears that the latter dominates the former, in the sense that it provided a better approximation quality, and was computed quicker for all considered instance parameters. It is particularily interesting that it was computed even faster than the *GrWLocR* greedy lower bound, which was obtained from greedily completing the solution of a uncoupled restriction with fewer variables, and therefore solved faster than the *restrict-split* variant. This gives us good reasons to believe that in applications, the latter will perform better than the former. However, this remains to be tested empirically.

In Appendix B, we present additionnal lower bound concepts, which are based on correcting upper bound solutions obtained by solving either the CS or the LR relaxations. These corrections are performed according to greedy heuristics, removing in priority assignments which minimize the aggregated profit to weight ratio, until constraints of REF that were violated by the solutions to the relaxation become satisfied again.

In the next sections, we investigate ways in which the DP approach to solving REF can be improved, using, among other devices, decomposition in independent decision sequences, incumbent sets and singleton upper bounds. In Section C.4.3, we propose improvements over the basic DP method which we characterize as "Bottom-up" approaches, based on using decomposition in the treatment of local tasks first, and delaying as much as possible the consideration of all residual capacity criteria at once. In Section C.4.4, we tackle choices associated with complex tasks first, so as to decouple the problem entirely before solving instances defined by these decoupling choices. Hence we characterize those approaches as "top-down".

4.3 "Bottom-up" approaches based on Dynamic Programming

In this section, we present improvements of the DP algorithm which take advantage of the fact that some subsequences of steps in the decision process are independent. In our case, subsequences

of steps associated with local tasks from each subsystem. This can take several forms. First, the precomputation of an incumbent set from a decomposable restriction, which can be used in bound reasoning to eliminate partial solutions. Second, a preliminary filtering of each state of the decision sequence can be done considering only $p + 1$ criteria, by partitioning the set of partial solutions according to the site in which the last assignment has been made. Third, we can initialize the DP process with a *strong lower bound set* obtained from the decomposable restriction of the problem to local tasks, effectively ignoring $|S| - 1$ criteria in all decision steps associated with local tasks.

4.3.1 Filtering with a set of incumbent solutions

At each step of the decision sequence, partial solutions can be eliminated by comparing strong upper bounds to the values of their extensions to the values of already known solutions. This is similar to *fathoming a node* in branch and bound, where a subtree can be avoided if it is shown that an upper bound over all solutions in this subtree is dominated by an incumbent solution. If an incumbent solution is found to dominate the strong upper bound on extensions of a partial solution, then all the extensions of this partial solution will be dominated by the known incumbent solution. It is thus useless to pursue extensions of this partial solution, and we filter it out.

This procedure, akin to what was proposed by Figueira et al. (2013), is described in its application to REF by Algorithm 22. The better the precomputed incumbent solutions, the more powerful this filtering mechanism, in terms of the number of extensions eliminated by a successful test. However, the cost of computing the incumbent has to be taken into consideration, as it weighs on the overall performance.

4.3.2 Partial filtering

When generating new feasible solutions during the DP process, it is possible to filter subsets of these new solutions without considering all $|S|$ residual capacity criteria. In particular this is the case when two solutions which did not previously dominate each other are modified by assignments consuming resources in the same subsystem. Consider, for $t \in T$, $s \in S(t)$, $Q_t(s)$ the set of states where task t was assigned to a machine of site s.

Proposition 4.6. *for* $x, x' \in \tilde{\mathcal{E}}(Q_{t-1}, (f, \bar{w}))$ *such that* $\tau(x,\delta), \tau(x',\delta') \in Q_t(s)$,

$$(f,\bar{w})(\tau(x,\delta)) \preceq (f,\bar{w})(\tau(x',\delta')) \text{ if and only if } (f,\bar{w}_s)(\tau(x,\delta)) \preceq (f,\bar{w}_s)(\tau(x',\delta'))$$

Proof. Left to right is trivial. For right to left, consider that if $\tau(x,\delta), \tau(x',\delta') \in Q_t(s)$, then for all $s' \neq s, \bar{w}_{s'}(\tau(x,\delta)) = \bar{w}_{s'}(x)$ and $\bar{w}_{s'}(\tau(x',\delta')) = \bar{w}_{s'}(x')$. In other words, on the one hand, if $\bar{w}_s(\tau(x,\delta)) \preceq \bar{w}_s(\tau(x',\delta'))$, then $\bar{w}(\tau(x,\delta)) \preceq \bar{w}(\tau(x',\delta'))$, and on the other hand, if $(f(\tau(x,\delta)), \bar{w}_s(\tau(x,\delta)) \preceq (f(\tau(x',\delta')), \bar{w}_s(\tau(x',\delta')))$, then in particular $f(\tau(x,\delta)) \preceq f(\tau(x',\delta'))$. The conjunction of these two facts yields the result. □

Algorithm 22: DP algorithm for solving REF with filtering using a set of incumbent solutions and a strong upper bounds on extensions of partial solutions.

input : $(T_C, T_L, M, S), I$

/*I denotes the precomputed incumbent set */

output: $\mathcal{N}(f(X))$

$Q_0 \leftarrow \{e_N\}$

for $t \in \{1, ..., \tilde{t}\}$ **do**

  for $x \in Q_{t-1}$ **do**

    $u^x \leftarrow$ a strong upper bound on $Ext(x, T)$.

    if $\neg\exists x' \in I$ *s.t.* $f(x') \geq u^x$ **then**

      for $m \in M(S(t))$ **do**

        $\tilde{x} \leftarrow x[x_{tm} \leftarrow 1]$

        if $\bar{w}(\tilde{x}) \geqq 0$ **then**

          $Q_t \leftarrow Q_t \cup \{\tilde{x}\}$

  if $t < \tilde{t}$ **then**

    $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, (f, \bar{w}))$

  else

    $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, f)$

return $Q_{\tilde{t}}$

This observation allows the following modification of the DP procedure, described as Algorithm 23, to produce the correct result. In essence this modification applies a sort of *divide and conquer* heuristic to filtering set of states Q_t, the latter being partitioned into $|S|$ sets of states, to which preliminary filterings are to be applied.

Algorithm 23: Partial filtering in the DP formulation of REF

input : (T_C, T_L, M, S)
output: $\tilde{\mathcal{E}}(X, f)$

for $t \in \{1, ..., \tilde{t}\}$ **do**
  for $x \in Q_{t-1}$ **do**
    for $s \in S(t)$ **do**
      for $m \in M(s)$ **do**
        $\tilde{x} \leftarrow x[x_{tm} \leftarrow 1]$
        if $\bar{w}(\tilde{x}) \geqq 0$ **then**
          $Q_t(s) \leftarrow Q_t(s) \cup \{\tilde{x}\}$
      $Q_t(s) \leftarrow \tilde{\mathcal{E}}(Q_t(s), (f, \bar{w}_s))$
    $Q_t \leftarrow \bigcup_{s \in S} Q_t$
  if $t < \tilde{t}$ **then**
    $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, (f, \bar{w}))$
  else
    $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, f)$
return $Q_{\tilde{t}}$

4.3.3 "Kickstarting"

Assume that T, which is both the set of tasks and the set of steps in the DP decision process, is ordered in such a way that all $t \in T_L$ come before all $t \in T_C$, and that t_C is the first complex task. Q_{t_C} can be computed by solving the subproblems associated with each $s \in S$ for criteria $(f, \bar{w}_s)$ and pooling the results together. Let l denote the last element of the set of local tasks, then according to Algorithm 16,

$$Q_{t_c} = \tilde{\mathcal{E}}(\{\tau(x, m) \in X_{t_c} \mid x \in Q_l, m \in M(S(t_c)) \cup e^{t_C}\}, (f, \bar{w}))$$

What we propose is a way to compute Q_l using decomposition. The subsequence of the DP decision process associated with T_L can be seen as a DP process where both the intermediary and target objective function are $(f, \bar{w})$. Thus Proposition C.3 implies that

$$\mathcal{N}((f, \bar{w})(X_L)) = (f, \bar{w})(Q_l)$$

X_L is exactly the feasible set of the *local restriction* of REF, and we know from section 3.2.2 that $X_L = \prod_{s\in S} X_L^s$. Again by Proposition C.3, for any $s \in S$, $T_L(s) = \{1^s, ..., l^s\}$ the set of local tasks associated with site s, we have that

$$\mathcal{N}((f, \bar{w})(X_L^s)) = (f, \bar{w})(Q_{l^s})$$

Algorithm 24: "Kickstarted" dynamic programming for solving REF

input : (T_C, T_L, M, S)
output: $\tilde{\mathcal{E}}(X, f)$

for $s \in S$ **do**
  $Q_{l^s} \leftarrow DP(\varnothing, T_L(s), M(s), \{s\})$
$Q_{t_C} \leftarrow \mathcal{E}(\prod_{s\in S} Q_{l^s}, (f, \bar{w}))$
/*Assume $T_C = \{t_C, ..., \tilde{t}\}$ */
for $t \in \{t_C, ..., \tilde{t}\}$ **do**
  for $x \in Q_{t-1}$ **do**
    for $m \in M(S(t))$ **do**
      $\tilde{x} \leftarrow \tilde{x}[x_{tm} \leftarrow 1]$
      if $\bar{w}(\tilde{x}) \geqq 0$ **then**
        $Q_t \leftarrow Q_t \cup \{\tilde{x}\}$
  if $t < |T_C| - 1$ **then** $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, (f, \bar{w}))$
  else $Q_t \leftarrow \tilde{\mathcal{E}}(Q_t, f)$
return $Q_{|T_C|}$

4.3.4 Experimental Results

In experiments reported in Tables 4.4 to 4.6, we measure the performance improvement yielded by implementing the modifications of the DP algorithm which we described in the previous subsections. The implementation of partial filtering (denoted by *Pt.Filt*) in these experiments does not require additional clarifications.

However, to apply the filtering of partial solutions using an incumbent set (denoted *LB.Filt*), as described in Section C.4.3.1, we first need to choose among of the available notions of strong upper bounds and weak lower bounds to constitute our incumbent. Our chosen notion of strong upper bound over the extensions of a partial solution will be the greedy singleton upper bound which we described in Section C.4.2.2 and proven to be an strong upper bound in Proposition C.4. We used the two types of lower bound which we have already experimented with in Section 4.2.3. Namely, the *GrS LocR*α lower bound, and the *RS* lower bound, which is the solution of the *restrict-split* restriction.

p	$\|M(s)\|$	$\|T_C\|$	$\|S\|$	$\|N\|$	*T. DP*		*LB Time*				*T. DP LB.Filt*			
					Basic		*Greedy LB*		*RS LB*		*Greedy LB*		*RS LB*	
2	4	2	2	48	0.434	[0.273]	0.014	[0.010]	0.021	[0.018]	0.343	[0.249]	0.220	[0.124]
2	5	2	2	70	3.316	[2.262]	0.037	[0.026]	0.035	[0.008]	0.942	[0.493]	0.822	[0.589]
2	6	2	2	96	12.896	[8.228]	0.089	[0.053]	0.088	[0.025]	2.594	[1.754]	1.950	[1.500]
2	4	3	2	56	1.390	[0.521]	0.014	[0.007]	0.015	[0.006]	0.638	[0.300]	0.600	[0.317]
2	4	4	2	64	2.420	[0.792]	0.014	[0.005]	0.034	[0.032]	0.808	[0.465]	0.593	[0.305]
2	4	5	2	72	5.299	[2.085]	0.021	[0.019]	0.028	[0.009]	2.200	[1.281]	1.217	[0.646]
2	4	2	3	72	20.940	[11.469]	0.027	[0.022]	0.018	[0.002]	4.884	[5.025]	2.457	[1.179]
2	3	1	4	48	5.010	[1.954]	0.010	[0.012]	0.005	[0.001]	1.053	[0.651]	0.870	[0.659]
2	3	2	4	60	35.278	[14.205]	0.011	[0.007]	0.007	[0.002]	2.848	[1.591]	2.258	[1.220]
3	5	2	2	70	8.633	[3.488]	0.087	[0.027]	0.087	[0.031]	3.504	[1.524]	4.259	[0.824]
3	5	3	2	80	28.138	[17.006]	0.316	[0.131]	0.158	[0.041]	14.468	[7.539]	11.951	[5.936]
3	4	2	3	72	136.061	[115.563]	0.131	[0.058]	0.040	[0.009]	29.965	[13.391]	33.937	[16.313]

Table 4.4: Filtering at each step using bound reasoning with the greedy UB over extensions of partial solutions, and, as LB, either greedy extensions of LocRes solutions or the RS lower bound. Average values for 10 trials.

p	$\|M(s)\|$	$\|T_C\|$	$\|S\|$	$\|N\|$	$\|ND\|$		*T. DP*			
							Basic		*Pt.Filt*	
2	4	2	2	48	17.950	[8.056]	0.434	[0.273]	0.380	[0.206]
2	5	2	2	70	23.150	[7.107]	3.316	[2.262]	2.702	[1.948]
2	6	2	2	96	39.050	[10.737]	12.896	[8.228]	12.327	[8.504]
2	4	3	2	56	19.100	[6.718]	1.390	[0.521]	1.145	[0.382]
2	4	4	2	64	24.800	[5.877]	2.420	[0.792]	2.004	[0.810]
2	4	5	2	72	29.350	[10.143]	5.299	[2.085]	4.712	[2.063]
2	4	2	3	72	25.500	[11.826]	20.940	[11.469]	20.503	[12.113]
2	3	1	4	48	11	[5.560]	5.010	[1.954]	4.392	[1.847]
2	3	2	4	60	25	[11.152]	35.278	[14.205]	33.487	[14.392]
3	5	2	2	70	173.800	[57.036]	8.633	[3.488]	8.835	[3.484]
3	5	3	2	80	467.800	[278.135]	28.138	[17.006]	28.375	[15.177]
3	4	2	3	72	203	[113.837]	136.061	[115.563]	157.179	[154.695]

Table 4.5: Using partial filtering over extensions built from tasks assignments in the same site, at each step of the DP process. Average values for 10 trials.

Results reported in table 4.4 show that our DP algorithm benefits most from using filtering using the *RS* lower bound as incumbent. Although it was in several cases more costly to compute, it led in all cases to a quicker resolution of the problem, which, in the biobjective case, led to solving the problem up to 15.6 times faster than the basic DP algorithm, and about 4 times faster for $p = 3$ on the tested instances. Thus, filtering using an incumbent set obtained from a weak lower bound seems to be a very reliable way to improve the performance of the DP based approach.

Experiments on partial filtering presented in Table 4.5 showed that, for $p = 2$, this trick provides a slight but systematic improvement of average performance. For $p = 3$ however we did not see an advantage to partial filtering. In any case, the average improvement yielded by partial filtering was smaller than the standard deviation and thus probably not reliable in practice. We believe there may be ways of further improving this idea by keeping in memory, from the first partial dominance test, which dominance tests will lead to an incomparability when all partitions of the set of states are pooled together for the main dominance filtering.

Finally, experiments reported in Table 4.6 reveal that kickstarting the DP process by initializing it with the *strong LocRes* lower bound computed by decomposition (the computation of this

p	$\|M(s)\|$	$\|T_C\|$	$\|S\|$	$\|N\|$	*T. DP* *Basic*		*T. DP. KS* *Basic*		*T. DP. KS* *LB.Filt*	
2	4	2	2	48	0.434	[0.273]	0.343	[0.154]	0.251	[0.122]
2	5	2	2	70	3.316	[2.262]	2.289	[0.937]	1.288	[0.530]
2	6	2	2	96	12.896	[8.228]	7.690	[1.734]	3.139	[0.876]
2	4	3	2	56	1.390	[0.521]	0.924	[0.596]	0.754	[0.400]
2	4	4	2	64	2.420	[0.792]	2.006	[0.791]	1.449	[0.467]
2	4	5	2	72	5.299	[2.085]	5.163	[1.197]	3.371	[0.945]
2	4	2	3	72	20.940	[11.469]	20.942	[9.059]	14.681	[6.392]
2	3	1	4	48	5.010	[1.954]	2.775	[0.866]	2.020	[0.647]
2	3	2	4	60	35.278	[14.205]	26.662	[12.918]	23.506	[11.313]
3	5	2	2	70	8.633	[3.488]	6.072	[2.047]	4.357	[1.276]
3	5	3	2	80	28.138	[17.006]	27.498	[12.885]	27.259	[12.702]
3	4	2	3	72	136.061	[115.563]	119.844	[59.523]	114.802	[56.841]

Table 4.6: "Kickstarted" dynamic programming algorithm, with bound reasoning and partial filtering improvements. Average values for 10 trials.

p	$\|M(s)\|$	$\|T_C\|$	$\|S\|$	*T. e − const* *NoDec*		*T. DP* *Basic*		*T. DP LB.Filt* *RS LB*		*T. DP. KS* *LB.Filt*	
2	4	2	2	0.640	[0.541]	0.434	[0.273]	0.220	[0.124]	0.251	[0.122]
2	5	2	2	0.976	[0.536]	3.316	[2.262]	0.822	[0.589]	1.288	[0.530]
2	6	2	2	2.519	[0.788]	12.896	[8.228]	1.950	[1.500]	3.139	[0.876]
2	4	3	2	0.662	[0.352]	1.390	[0.521]	0.600	[0.317]	0.754	[0.400]
2	4	4	2	0.846	[0.340]	2.420	[0.792]	0.593	[0.305]	1.449	[0.467]
2	4	5	2	1.333	[0.818]	5.299	[2.085]	1.217	[0.646]	3.371	[0.945]
2	4	2	3	1.314	[0.780]	20.940	[11.469]	2.457	[1.179]	14.681	[6.392]
2	3	1	4	0.317	[0.275]	5.010	[1.954]	0.870	[0.659]	2.020	[0.647]
2	3	2	4	1.006	[0.657]	35.278	[14.205]	2.258	[1.220]	23.506	[11.313]
3	5	2	2	19.534	[5.435]	8.633	[3.488]	4.259	[0.824]	4.357	[1.276]
3	5	3	2	60.296	[31.120]	28.138	[17.006]	11.951	[5.936]	27.259	[12.702]
3	4	2	3	27.269	[16.142]	136.061	[115.563]	33.937	[16.313]	114.802	[56.841]

Table 4.7: Comparing DP-based approaches to solving REF. Average values for 10 trials.

bound being of course counted in the computing time for *KS DP*) does allow some performance improvement. We achieved additional performance improvements by combining kickstarting with the use of filtering using the RS LB as set of incumbent solutions, and applying filtering both in the local subproblems and when assigning complex tasks. Kickstarted DP works especially well when the number of complex tasks is small in proportion of the number of local tasks. However, in the opposite case, it appears to perform very similarly to the basic DP algorithm, especially so when the number of criteria increases.

Table 4.7 summarizes the results obtained in this subsection, to determine the best "bottom-up" method available. The clear winner is the approach filtering with the RS LB incumbent set over the basic DP sequence, rather than the kickstarted one.

This may be surprising at first, but to understand it, one may note that occurrences of filtering of partial solutions using the incumbent set in the decoupled subproblems of the KS approach were virtually inexistent. Indeed, as partial solutions receive assignments, the upper bound over their extensions becomes tighter. In a short experiment reported in Table 4.8 we consider an instance with $|M(s)| = |T_L(s)| = 4$, $|S| = 3$ and $|T_C| = 2$. We solved the basic, unisequential DP algorithm with filtering using the *RS LB*. Elimination almost never happened before step 5, and local subproblems only assign 4 tasks. Partial solutions of subproblem simply do not go through

Stp idx	1	2	3	4	5	6	7
# El	0 [0]	0 [0]	0 [0]	0.4 [0.5]	9.4 [6.9]	65.6 [44.6]	90.2 [44.6]
Stp idx	8	9	10	11	12	13	14
# El	187.6 [60.0]	345.4 [146.9]	569.4 [331.7]	799 [434.7]	1538.8 [984.1]	1840.2 [1182.2]	1865.2 [1050.8]

Table 4.8: Number of partial solutions eliminated at each step of the basic DP process, using greedy UB and *RS LB* incumbent set. Average values for 5 trials.

enough assignments for the upper bound dominance test to succeed.

Further work is required for filtering using an incumbent set in the kickstarted approach, using an upper bound that can eliminate partial solutions in earlier steps of the decision process. We have observed that using the objective-wise optimum of the LP relaxation of the completion problem tends to eliminate partial solutions earlier (and fewer in later stages), but it appeared too costly in preliminary experiments. Yet, it hints at a possible way of improving this method.

To summarize, all dynamic programming-based approaches suffer major impairments when $|S|$ increases. This reflects the fact that an increasing $|S|$ generates more incomparabilities between partial solutions, which increases the number of partial solutions to be handled until the very last stage. This remains the main flaw of the "bottom-up" approach to solving REF, which we have not yet found a way to overcome. As a consequence, we will now explore methods which use DP when only one residual capacity criterion is involved, i.e. when the problem is reduced to a collection of *restrict-split* variants, as will be investigated in the last section of this work.

In the next section, we shift the perspective on solving the REF problem. We started by solving it from the bottom up, starting from local task assignments and keeping sufficient residual capacity to assign complex tasks. Next, we will investigate how preliminary coordination of the assignment of complex task may allow for a faster resolution of REF than that available with generic methods.

4.4 "Top-down" approaches for solving REF

A central hypothesis in the decomposition approach to the optimization of complex systems is that a collection of independent subsystem problems is significantly easier to solve than the whole problem. If this discrepancy is sufficiently large, in particular if subproblems can be solved using methods which specifically exploit their structural properties to reach higher efficiency, it may be possible to solve not just the collection of subproblems associated with one variant, but several of such collections, and in less time than a generic algorithm requires to solve the whole original problem. Under some conditions, solving such collections of variants can lead to an exact solution to the original MOIP problem.

4.4.1 Enumerating all ways of decoupling REF

Recall from Section 3.2.2 that the *restrict-split* variant of any instance of the *GCP* is defined by assigning each coupling constraint to some subsystem $s \in S$, yielding subset $K_s \subseteq K$ of coupling constraints involving variables from subsystem s, and in which we keep only variables x_s free. The restriction is thus formulated as

$$\begin{array}{lll} \max & f(x_{\bar{K}}, x_{K_f}, e_{\overline{K_f}}) & \\ s.t. & (x_k^s, e_k^{\bar{s}}) \in X^k & \forall s \in S, \forall\, k \in K_s \\ & x^s \in X^s & \forall\ s \in S \end{array}$$

or equivalently,

$$\begin{array}{lll} \max & f(x_{\bar{K}}, x_{K_f}, e_{\overline{K_f}}) & \\ s.t. & (x_{\bar{K}}^s, x_{K_s}^s) \in X_s' & \forall s \in S \end{array}$$

where $X_s' = X^s \cap \bigcap_{k \in K_s} X_k'$, and X_k' is a set such that $x_k^s \in X_k'$ if and only if $(x_k^s, e_k^{\bar{s}}) \in X^k$.

Note that the set S^K of applications of K into S defines the set of all possible assignments of couplings constraints to subsystems. Thus to each $\sigma \in S^K$ corresponds one *restrict-split* variant, and for X^σ the feasible set of any such variant, we have that $X^\sigma \subseteq X$. If we can prove, for a particular coupled problem, that $\bigcup_{\sigma \in S^K} X^\sigma = X$, then the following fact ensures that we can solve the original problem by solving all possible *decoupled variants* of the problem.

Observation 4.2. *Let $(X^i \,|\in I)$ be a family of sets with $X^i \subseteq X$ for each $i \in I$, and $f : X \to \mathbb{R}^p$. If $\bigcup_{i \in I} X^i = X$, then*

$$\mathcal{E}(\bigcup_{i \in I} \mathcal{E}(X, f)) = \mathcal{E}(X, f)$$

or equivalently for $(Y^i \mid i \in I)$ with $Y^i \subseteq Y \subseteq \mathbb{R}^p$ for each $i \in I$, if $\bigcup_{i \in I} Y^i = Y$, then

$$\mathcal{N}(\bigcup_{i \in I} \mathcal{N}(Y^i)) = \mathcal{N}(Y)$$

We consider the case of REF, with $\sigma \in S^{T_C}$, the restrict split restriction of which is formulated as follow. Further on, X^σ will denote the feasible set of a restrict split variant of REF for σ.

$$
\begin{aligned}
\max \quad & \sum_{t\in T}\sum_{m\in M(S(t))} g^j_{tm}x_{tm} && \forall\, j\in\{1,...,p\}\\
s.t. \quad & \sum_{m\in M(S(t))} x_{tm} \leqslant 1 && \forall\, t\in T\\
& \sum_{t\in T(s)}\sum_{m\in M(S(t))} w_{tm}x_{tm} \leqslant b_s && \forall\, s\in S\\
& x_{tm}\in\{0,1\} && \forall\, t\in T,\ m\in M(S(t))\\
& x_{tm}=0 && \forall\, t\in T_C, m\notin M(\sigma(t))
\end{aligned}
$$

Proposition 4.7. *For some instance of REF with feasible set X and $\sigma \in S^{T_C}$ defining a restrict split variant of REF with feasible set X^σ, $\bigcup_{\sigma\in S^K} X^\sigma = X$*

Proof. $\subseteq$: This follows straightforwardly from the fact that 0 is an admissible value for x_{tm}.

$\supseteq$: Let $x \in X$. For each t, either for all $m \in M(S(t))$, $x_{tm} = 0$, or not. If not, then consider $\bar{t}$ and $m \in M(\bar{t})$ such that $x_{\bar{t}m} = 1$. Then, $x \in X$ implies that for all $m' \neq m$, $x_{tm'} = 0$. Thus for each $t \in T_C$, there is a unique $s \in S$ such that for all $m \notin M(s)$, $x_{tm} = 0$. This means that for each x, there is some $\sigma \in S^{T_C}$ such that $x \in X^\sigma$. □

Thus we can solve an instance of REF with the simple procedure described by Algorithm 31:

Algorithm 25: Resolution algorithm enumerating all decouplings of REF

```
Out ← ∅
for σ ∈ S^{T_C} do
    Out ← Out ∪ Ẽ(X^σ, f)
return Ẽ(Out)
```

To summarize, decoupling a problem amounts to fixing some variables to admissible values, effectively turning coupling constraints into non-coupling, local constraints. Using notation previously introduced for restrict-split variants of coupled problems, we saw that the set S^K of functions σ associating each coupling to a particular subsystem describes an exhaustive exploration of all ways of decoupling the original problem.

Tables 4.9 and 4.10 present results of an experiment where an instance of *REF* is solved by enumerating all of S^{T_C} (since $K = T_C$ in this case). Instances are such that each site has the same number $|M(s)|$ of machines. For each $\sigma \in S^{T_C}$ is solved as a whole (*NoDec*), and then enumerating all restrict-split variants (*Enum*), which are solved by decomposition, subproblems being solved using e-constraint method for $p = 2$, and the generic algorithm by Tamby and Vanderpooten (2021) for $p \geqslant 3$. This experiment shows that even in the biobjective case, enumeration of S^{T_C} (column *Enum&Dec*) can be competitive and even faster than generic resolution when the number of subsystems is relatively low compared to the size of subsystems. However, for smaller subsystems but a larger value of S^{T_C}, this approach performs worse than the generic method.

p	$\lvert M(s)\rvert$	$\lvert T_L(s)\rvert$	$\lvert T_C\rvert$	$\lvert S\rvert$	$\lvert N\rvert$	$\lvert S^{T_C}\rvert$	$\lvert ND\rvert$	$T.e-const$			
								NoDec		*Enum&Dec*	
2	5	5	2	2	70	4	25.725	[10.08]	0.826	[0.405]	0.713	[0.233]
2	6	5	2	2	84	4	32.25	[10.539]	1.031	[0.441]	0.866	[0.302]
2	6	6	2	2	96	4	37.775	[13.339]	1.688	[0.696]	1.425	[0.484]
2	7	6	2	2	112	4	41.05	[15.717]	1.609	[0.707]	1.535	[0.625]
2	7	7	2	2	126	4	53.775	[15.94]	2.415	[1.138]	2.248	[0.816]
2	5	5	3	2	80	8	27.925	[10.088]	0.946	[0.372]	1.639	[0.536]
2	5	5	2	3	105	9	44.525	[13.478]	2.123	[0.934]	1.904	[0.488]
3	5	5	2	2	70	4	203.325	[106.873]	23.567	[12.061]	7.699	[3.419]
4	4	4	2	2	48	4	234.9	[122.423]	45.449	[30.068]	14.632	[4.992]

Table 4.9: Instances with few "big" subsystems (average time for 10 trials)

p	$\lvert M(s)\rvert$	$\lvert T_L(s)\rvert$	$\lvert T_C\rvert$	$\lvert S\rvert$	$\lvert N\rvert$	$\lvert S^{T_C}\rvert$	$\lvert ND\rvert$	$T.e-const$			
								NoDec		*Enum&Dec*	
2	2	2	2	4	32	16	9.075	[5.47]	0.096	[0.084]	0.47	[0.045]
2	2	2	1	6	36	6	8.425	[5.254]	0.06	[0.047]	0.216	[0.021]
2	3	3	2	4	60	16	21.675	[10.047]	0.565	[0.253]	0.77	[0.11]
2	3	3	1	5	60	5	18.625	[8.72]	0.398	[0.249]	0.236	[0.031]
3	3	3	2	4	60	16	131.475	[72.204]	10.293	[6.849]	1.929	[0.647]
4	3	3	2	4	60	16	483.6	[264.622]	117.289	[83.511]	37.164	[5.005]

Table 4.10: Greater number of 'small' subsystems (average time for 10 trials)

4.4.2 Pre-computation of independent dynamic programming states

The restrict-split variant can be solved by decomposition, and each of the subsystem problems of this variant can be solved by DP. Thus, solving all possible restrict-split restrictions of the original problem boils down to solving only DP subproblems with $p+1$ criteria. By definition, these subproblems are restricted to local tasks of some subsystem of the original formulation, in addition to the new local tasks obtained by assigning complex tasks to a particular subsystem according to $\sigma \in S^{T_C}$.

Since most of the tasks in the original problem are not complex but local tasks, their treatment will be the same in each node. It is well known that the output of a DP algorithm does not depend on the order in which the decisions are taken. Without loss of generality, we can assume that local tasks are always taken care of first, so that the $\lvert T_L\rvert$ first steps of the resolution of decoupled problems are the same in each leaf-node of the decoupling branching scheme. Thus, these steps can be precomputed: each of the subsystem specific subproblem in a leaf-node problem could be kickstarted to the first task step corresponding to a complex tasks having been assigned to this subsystem in the restrict-split variant. But contrary to the kickstarted method considered in Section C.4.3.2, we now solve problems with only one residual capacity criterion.

We recall that $T_L(s) = T(s) \cap T_L$, and we consider the feasible set X_L^s of subproblem s restricted to local task variables, defined as :

$$X_L^s := \{x \in \{0,1\}^{|T_L(s)|} \mid \sum_{t \in T_L(s)} \sum_{m \in M(s)} x_{tm} \leqslant 1 \quad \& \sum_{t \in T_L(s)} \sum_{m \in M(s)} w_{tm} x_{tm} \leqslant b_s\} \tag{4.1}$$

Algorithm 26 describes the computation of $\mathcal{N}(f(X))$. First, we compute $Q_L^s = \tilde{\mathcal{E}}(X_L^s, (f^s, \bar{w}_s))$

for all $s \in S$, i.e. the solutions of the restriction of subproblems to local tasks which are non-dominated for f and for $\bar{w}_s$, the residual capacity criterion local to their subproblem. Then for each $\sigma \in S^{T_C}$, we complete the resolution of the $|S|$ independent subproblem with the tasks made local to each of them by decoupling σ, and we compute the non-dominated subset of the cartesian product of these solutions. Since the restrict-split problem has no coupling, this yields all the efficient solutions of this decoupled version of the problem. We then compute the union of all the non-dominated sets obtained in restrict-split variants, and the non-dominated subset of this union is $\mathcal{N}(f(X))$.

Algorithm 26: Resolution algorithm enumerating all decouplings of REF with pre-computation of local steps)

$Q_L^s \leftarrow \tilde{\mathcal{E}}(X_L^s, (f^s, \bar{w}_s))$ for each $s \in S$
$Out \leftarrow \varnothing$
for $\sigma \in S^{T_C}$ **do**
    $Out^\sigma \leftarrow \varnothing$
    for $s \in S$ **do**
        $Out^s \leftarrow DP(Q_L^s, \varnothing, T^\sigma(s), M(s))$
        /*DP is initialized with precomputed state Q_0^s */
        $Out^\sigma \leftarrow \tilde{\mathcal{E}}(Out^\sigma \times Out^s)$
    $Out \leftarrow Out \cup Out^\sigma$
return $\mathcal{E}(Out)$

4.4.2.1 Experimental Results

Table 4.11 reports $|S^{T_C}|$ the total number of decouplings, $|ND|$ the size of the non-dominated set, *T.Enum.DP* the computing time of the enumerative method where the subproblems of each restrict-split decoupling are solved using the *DP*, and *T.Enum.DP.PreC* the enumerative method where the local parts are precomputed and passed to subproblems, at which point the DP procedure is merely *completed* for remaining tasks.

As is apparent from results in Table 4.11, the joint use of dynamic programming and the pre-computation of the DP subsequence which is common to all restrict-split variants leads to very significant performance improvements. In the bi-objective case, for weakly coupled instances, we solved some instances of the problem 4 to 8 times faster than the generic algorithm, and for $p = 3$, we find 18 to 52-fold speed-ups relative to the generic method.

Finally, we compare the performance of the best methods obtained in Section C.4.3 with the best "top-down" approach. Considering first instances solved in the previous section, we observe, from results presented in Table 4.12 that in all cases, the top-down approach denoted *Dec DP PreC* performs significantly better than the bottom-up approach denoted *DP LB.Filt*.

p	$\|M(s)\|$	$\|T(s)\|$	$\|T_C\|$	$\|S\|$	$\|S^{T_C}\|$	$\|N\|$	$\|ND\|$		$T.e-const$		$T.Enum.\ DP$		$T.EnDPPrec$	
2	5	5	2	2	4	70	22.3	[8.48]	0.692	[0.283]	0.607	[0.227]	0.132	[0.028]
2	6	6	2	2	4	96	38.7	[14.37]	1.495	[0.679]	1.211	[0.433]	0.452	[0.121]
2	4	4	2	3	9	72	23.55	[5.728]	0.87	[0.305]	0.763	[0.253]	0.151	[0.029]
2	3	3	3	3	27	54	21.3	[8.542]	0.684	[0.399]	1.344	[0.336]	0.157	[0.044]
3	4	4	2	2	4	48	78.5	[30.299]	5.256	[2.156]	1.413	[0.614]	0.075	[0.031]
3	4	5	2	2	4	56	128.2	[74.695]	11.943	[8.067]	3.705	[2.254]	0.216	[0.133]
3	4	4	2	3	9	72	213.1	[103.515]	20.346	[11.787]	4.331	[1.11]	0.387	[0.098]

Table 4.11: Computing time (in seconds) for solving the global problem with a generic method ($T.e-const$), solving all restrict split variants with DP ($T.\ Enum\ DP$), and solving restrict split variants with DP and using a pre-computation of shared subsequence ($T.EnDPPrec$). Average values for 10 trials.

p	$\|M(s)\|$	$\|T_C\|$	$\|S\|$	$\|N\|$	$\|S^{T_C}\|$	$\|ND\|$		$T.\ e-const$ $NoDec$		$T.\ DP\ LB.Filt$ $RS\ LB$		$T.\ Enum$ $Dec\ DP\ PreC$	
2	4	2	2	48	4	17.950	[8.056]	0.640	[0.541]	0.220	[0.124]	0.026	[0.009]
2	5	2	2	70	4	23.150	[7.107]	0.976	[0.536]	0.822	[0.589]	0.146	[0.059]
2	6	2	2	96	4	39.050	[10.737]	2.519	[0.788]	1.950	[1.500]	0.319	[0.171]
2	4	3	2	56	8	19.100	[6.718]	0.662	[0.352]	0.600	[0.317]	0.130	[0.061]
2	4	4	2	64	16	24.800	[5.877]	0.846	[0.340]	0.593	[0.305]	0.462	[0.136]
2	4	5	2	72	32	29.350	[10.143]	1.333	[0.818]	1.217	[0.646]	0.794	[0.254]
2	4	2	3	72	9	25.500	[11.826]	1.314	[0.780]	2.457	[1.179]	0.208	[0.063]
2	3	1	4	48	4	11	[5.560]	0.317	[0.275]	0.870	[0.659]	0.027	[0.013]
2	3	2	4	60	16	25	[11.152]	1.006	[0.657]	2.258	[1.220]	0.155	[0.082]
3	5	2	2	70	4	173.800	[57.036]	19.534	[5.435]	4.259	[0.824]	0.430	[0.183]
3	5	3	2	80	8	467.800	[278.135]	60.296	[31.120]	11.951	[5.936]	1.032	[0.339]
3	4	2	3	72	9	203	[113.837]	27.269	[16.142]	33.937	[16.313]	0.585	[0.383]

Table 4.12: Comparing computing time (in seconds) with the best bottom-up approach and the best top-down approach. Average values for 10 trials.

p	$\|M(s)\|$	$\|T_C\|$	$\|S\|$	$\|N\|$	$\|S^{T_C}\|$	$\|ND\|$		$T.\ e-const$ $NoDec$		$T.\ DP\ LB.Filt$ $RS\ LB$		$T.\ Enum$ $Dec\ DP\ Precomp$	
2	7	2	2	126	4	51.700	[18.723]	2.752	[1.216]	6.849	[3.758]	0.636	[0.175]
2	8	2	2	160	4	63	[10.745]	3.155	[0.432]	12.406	[3.793]	1.123	[0.181]
2	9	2	2	198	4	107.800	[39.524]	6.532	[3.688]	63.682	[51.077]	3.295	[1.214]
2	4	3	3	84	27	41.400	[21.188]	1.656	[0.843]	6.705	[4.812]	0.647	[0.151]
2	5	3	3	120	27	51.200	[12.127]	2.690	[0.766]	23.678	[10.855]	1.382	[0.280]
2	4	4	3	96	81	45.800	[15.284]	2.960	[1.427]	26.657	[17.241]	2.150	[0.643]
2	5	4	3	135	81	70.500	[17.079]	5.480	[3.206]	116.698	[83.236]	7.241	[1.799]
2	3	4	4	84	256	39.500	[14.354]	2.104	[1.191]	42.145	[34.344]	3.134	[0.539]
2	4	4	4	128	256	62.300	[21.617]	5.285	[1.865]	445.941	[419.369]	11.060	[2.721]
3	6	2	2	96	4	481.500	[203.468]	65.729	[27.388]	14.297	[7.151]	1.108	[0.497]

Table 4.13: Comparing computing time (in seconds) with the best bottom-up approach and the best top-down approach, on harder instances. Average values for 10 trials.

We further considered "harder" instances, where we increased either the number of local tasks and machines $|M(s)|$, or the degree of coupling between the subproblems, with more subsystems and more complex tasks. The superiority of the top-down approach, relative to the bottom-up approach, was confirmed by results obtained on these instances and reported in Table 4.13. When $|M(s)|$ increases, the top-down approach maintains an advantage over the $e-constraint$ method. However, for $p = 2$, when the degree of coupling increases, it is overcome, and $e-constraint$ remains the quicker method. In the next and last section, we pursue final improvements of the top-down approach aiming at staying competitive with $e-constraint$.

4.4.3 Towards a branch and bound method for solving coupled problems

4.4.3.1 Dominance between decouplings

We have seen that in some cases, the full enumeration of S^{T_C} is already competitive with a generic method. Thus, we can expect to reach good results by embedding it in a branch and bound approach, where branching is done not over the binary decision variables of the initial formulation, but over additional decision variables restricting a coupling to a particular subsystem. That being said, we remain wary of the two following facts:

1. a tree defined by branching over all possible restrictions of some coupling grows faster in width than a binary tree.
2. a leaf of the tree enumerating the values of decoupling variables does not define a solution because, it merely restrict the possible values of the variables of the original problem. Rather, it defines a decomposable instance of the problem, that is assumed to be solvable quickly.

Single objective approaches to the optimization of complex systems rely on the notion of coordination, i.e. of modifications of the subproblems which allow the independent resolution of these subproblems to yield a feasible solution to the global problem. Decouplings, which correspond to restrict-split variants in the generic form of the integer coupled optimization problem, consist in such coordinations. When solving single objective REF, we may consider the optimal value of an instance, and deduce the decoupled variant from which it is obtained. If for each instance, there is a best decoupling, that which is associated with the optimal value, then we may look for ways to either make a guess about which decoupling to pursue first, or to use branch and bound techniques to eliminate decouplings which we could prove would not lead to the optimal solution.

In the multiobjective case, what would a best decoupling mean? Conversely, would we be able to eliminate some decouplings in hopes to shorten the enumeration of S^{T_C}? Because we will compare sets of non-dominated points obtained by solving various restrict-split instances, let us define the notion of *set dominance*.

Definition 4.1. *Let $p \in \mathbb{N}$, $A, B \subseteq \mathbb{R}^p$. A dominates B if and only if for all $y \in B$, there exists some $y' \in A$ such that $y' \succeq y$.*

p	$\|M(s)\|$	$\|T(s)\|$	$\|T_C\|$	$\|S\|$	$\|N\|$	$\|S^{T_C}\|$	*Lvs dtd ND* (%)		*Lvs dtd Lvs* (%)	
2	4	3	3	4	96	64	88.906	[6.050]	82.969	[9.937]
2	4	3	4	3	84	81	89.383	[4.773]	84.444	[6.473]
2	3	3	4	4	84	256	96.836	[0.978]	95.117	[2.102]
2	3	2	4	5	90	625	98.224	[0.837]	95.552	[3.528]
2	4	4	5	4	144	1024	98.315	[0.373]	95.850	[1.436]
3	3	3	2	4	60	16	48.625	[14.432]	41.634	[18.246]
3	3	3	3	4	72	64	71.914	[10.503]	62.711	[14.929]

Table 4.14: Proportion of dominated decouplings as a function of various instance parameters. Average values for 40 trials.

Let $\sigma \in S^{T_C}$ identify one of the possible restrict-split instances, with feasible set denoted X^σ, a restriction of the original feasible set X. If it was the case that for all $\sigma \in S^{T_C}$, $\mathcal{N}(f(X^\sigma))$ dominates $\mathcal{N}(f(X^\tau))$, then σ would obviously be the best decoupling. However, this is extremely unlikely if the profits associated for each criterion with task assignments are uncorrelated.

Conversely, we can determine a decoupling $\sigma \in S^{T_C}$ to be undesirable *a posteriori*, either if it is dominated by $\mathcal{N}(f(X))$, or if there exists $\tau \in S^{T_C}$ such that $\mathcal{N}(f(X^\tau))$ dominates $\mathcal{N}(f(X^\sigma))$. The second case of course implies the first case by transitivity. In both case, we may be able to use multiobjective branch and bound reasoning to avoid enumerating all decouplings. However, if only the first case is met, because it may be that only the union of non-dominated sets associated with a subset of leaf-instances can dominate $\mathcal{N}(f(X^\sigma))$, a heavier computational effort will be required for a successful bound test.

In the following experiment, we will solve a number of instances and measure *a posteriori* the proportion of leaf instances such that their non-dominated set is dominated by the non-dominated set of the whole problem (column *Lvs dtd ND*), and proportions of leaf instances such that their non-dominated set is dominated by the non-dominated set of another leaf-instance (column *Lvs dtd Lvs*).

Results are reported in Table 4.14. We found, as is necessary, that *Lvs dtd Lvs* is always smaller than *Lvs dtd ND*, but *Lvs dtd ND* was surprisingly large, although profit coefficients were chosen randomly and uncorrelated, it appears that an overwhelming majority (up to about 96% in our experiments) of decouplings can be dominated by a single other decoupling.

This makes possible to consider a branch and bound (BB) approach to *REF* in which we would branch over the decisions to assign complex tasks to sites, rather than to machines. Thus, we now present branch and bound (BB) notions directly in the multiobjective case, and in a way that will allow us to easily introduce the unusual 'decoupling" branching scheme particularily suited for decomposition.

4.4.3.2 Arborescent search and branching

Let $\mathcal{T} \subseteq \mathcal{P}(\mathbb{N})$ be a set of subsets of variables indices. These will define, for each level of the BB tree, the subset of variables which will be fixed to some values. In general, $\mathcal{T}$ needs not define a partition of N, however it must be the case that $\bigcap_{t\in\mathcal{T}} t = \varnothing$, so that we do not assign values to variables more than once. Let $G = (V, E)$ be a tree of depth $|\mathcal{T}|$, where $v \in V$ is called a node. To each node is associated a restriction of the original optimization problem, with X_v the feasible set at node v. E is such that for all $v, v' \in V$, if $(v, v') \in E$ then $X_{v'} \subseteq X_v$. For each v in the tree such that $depth(v) = t$, for each v' such that $(v, v') \in E$, and u an evaluation of all decisions variables in t,

$$X_{v'} \in \{X \in \mathcal{P}(X_v) \mid \forall x \in X_v, \forall i \in t, x_i = u_i\}$$

In other words, each child of a given node of depth t has the subvector of variables corresponding to t fixed to some value. Let us now explain how these valuations are defined.

When formalizing a branch and bound resolution method with branching over subsets of variables, one can make use of additional variables subvectors x^t for $t \in T$ representing sets of variables. At each level $t \in T$, a branching decision must be made, i.e. an assignment of the variables x_i for $i \in t$. Borrowing notation from DP (cf. Section C.4.1.2), the set of branching decisions that can be made at level t will be written Δ_t, and $\tau(x, \delta)$ will be the solution obtained by modifying x according to decision δ, i.e. to each $\delta \in \Delta_t$ corresponds one assignments of the whole variable subvector $x^t = (x_i \mid i \in t)$. Thus, children node of node v of depth t, as defined by the feasible sets of their associated problems, are

$$\left\{X_{v'[x^t \leftarrow \tau(x^t,\delta)]} \mid \delta \in \Delta_t\right\} \text{ where } X_{v'[x^t \leftarrow a]} = \left\{x \in X_v \mid x^t = a\right\}$$

Example 4.4. *Consider a coupled problem with feasible set $X \subseteq$, S a set of subsystems, T_C a set of complex tasks. The partition $\mathcal{T}_C$ is defined so that each of its elements is the set of variables associated with a coupling. Thus $\mathcal{T}_C = \{\{(t, m) \mid m \in M(S(t))\} \mid t \in T_C\}$, i.e. each depth of the tree is associated with a complex task, and more precisely with the subset of variables coupled by the coupling. Let us define $\Delta_t = S(t)$ the set of branching decisions associated with each $t \in T_C$. Taking decision $s \in S(t)$ will be represented by $\tau(x^t, s) = x[x_{tm} \leftarrow 0 \mid m \in M(S(t))\backslash M(s)]$, i.e. all assignments of task t to machines which are not in the chosen s are set to 0. Then given node v at depth $t - 1$, children nodes v are defined by $\{X_{v^s} \mid s \in \delta_t\}$, with $X_{v^s} := \{x \in X_v \mid x^t = \tau(x^t, s)\}$. Figure C.13 contrasts this sort of* decoupling *branching with the binary branching that is classical in branch and bound.*

The decoupling branching scheme thus defined is not, in the case of REF, such that all variables are fixed in the problems associated with its leaf nodes. This is no problem *per se*, as long as we have a method to solve efficiently the leaf node instances. Because, in our cases, these instances are restrict-split variants of REF, we can solve them using decomposition, DP and precomputation of DP subsequences for subproblems, as previously described.

Two properties are of particular interest for BB resolution methods, exhaustivity and exclusiv-

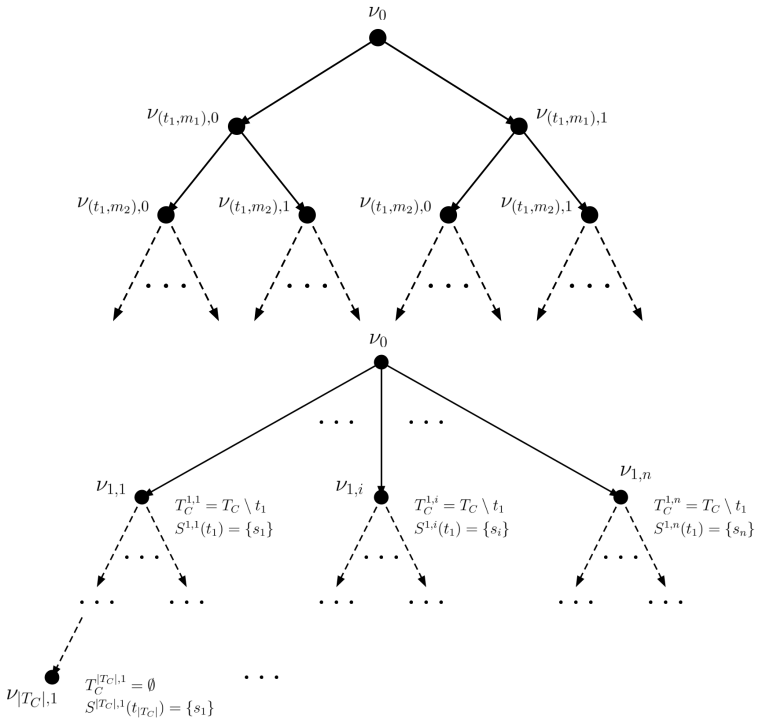


Figure 4.8: Binary branching (top) versus decoupling branching (bottom)

ity of branching, defined as follows:

Definition 4.2. *E is exhaustive if and only if for all* $\nu \in V$

$$\bigcup_{\nu' s.t. (\nu,\nu') \in E} X_{\nu'} = X_\nu \tag{4.2}$$

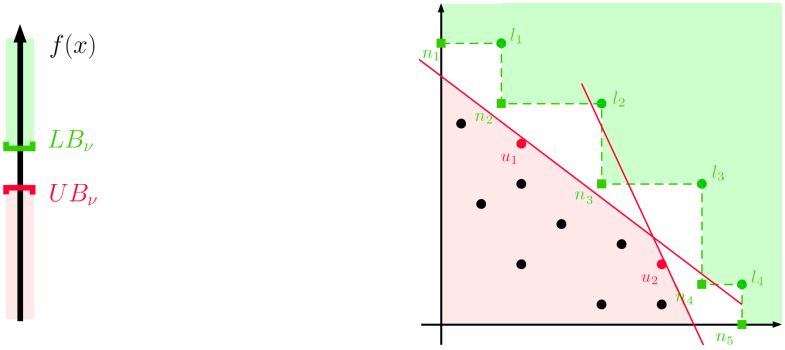
Definition 4.3. *E is exclusive if and only if for all* ν', ν'' *such that* $(\nu, \nu') \in E, (\nu, \nu'') \in E$

$$X_{\nu'} \cap X_{\nu''} = \varnothing \tag{4.3}$$

A BB algorithm is correct if the branching scheme is exhaustive. It needs not be exclusive, but exclusivity generally allows for better performance as it avoids redundancy. However, in cases where a particular branching scheme can yield a significantly smaller tree than usual exclusive branching schemes (e.g. branching over binary variable evaluations), with still sufficiently easy node-problems, it may be worth considering a non-exclusive branching scheme. This challenge is undertaken by designing sets of decisions which perform the adequate coordinations.

4.4.3.3 Fathoming a node in the branch and bound search

Let us introduce the implicit enumeration trick used in BB (without loss of generality we consider maximization). Consider $LB \subset \mathbb{R}^p$ a set of values of feasible solutions, such that for all $y \in LB$, there does not exist any $y' \in LB, y' \geq y$. At any given moment in the development of the method,



(a) Single objective case

(b) Biobjective case

Figure 4.9: The notions of search region, search zones and separation illustrated in single and biobjective cases. In case (b), search zones are defined by *knee* points n_1 to n_5, deduced from the lower bound $\{l_1, ..., l_4\}$ and some objective-wise upper bounds

LB (a *weak lower bound*) will be the set of the best solution values discovered so far, called the *incumbent* set. We note SR the subset of $\mathbb{R}^p$ in which there may still be solutions of interest to be found, i.e. solutions which are not dominated by elements of LB. Formally $SR = \{y \in \mathbb{R}^p \neg\exists y' \in L, y \preceq y'\} = (LB - \mathbb{R}^p_>)$, where $\mathbb{R}^p_> = \{x \in \mathbb{R}^p \mid x > 0\}$. This notion was introduced as the *search region* by Klamroth et al. (2015). If $p = 1$, $|LB| = 1$, so SR is simply the real half-line anchored in some $l \in \mathbb{R}$ such that $LB = \{l\}$. If $p \geqslant 2$, SR is defined as the union of subsets of $\mathbb{R}^p$ called *zones*. These are anchored in *knee* points defined by p-uples of incumbent points or partial solutions. They are the smallest set Z of points in $\mathbb{R}^p$ such that $\bigcup_{z\in Z}(\{z\} \oplus \mathbb{R}^p_>) = SR$.

The branch and bound principle is then the following: if one can prove that $f(X_\nu) \cap SR = \varnothing$, one shows that searching $G(\nu)$, the subtree rooted in node ν, cannot lead to the discovery of new non-dominated points. Therefore, ignoring this subtree will not prevent the enumeration of $\mathcal{N}(f(X))$. This is called a separation problem. Ehrgott and Gandibleux (2001), following Sourd and Spanjaard (2008), proposed to prove the absence of interesting solution in the subtree by enclosing $f(X_\nu)$ in an *upper bound set* U, thus proving that $(UB + \mathbb{R}^p_>) \cap SR = \varnothing$. This is illustrated in both single and multiobjective cases by Figure C.14.

For each node ν, let UB_ν be an upper bound set to $\mathcal{N}(f(X_\nu))$. Then by transitivity of dominance, it is an upper bound to $f(X_\nu)$. Since for all ν' with $(\nu, \nu') \in E$, it is the case that $X_{\nu'} \subseteq X_\nu$, we also have $f(X_{\nu'}) \subseteq f(X_\nu)$. Thus we know that no solution x found in the subtree rooted in ν can be such that $f(x) \in \{y \in \mathbb{R} \mid \exists y' \in UB_\nu, y \succ y'\} =: UB + \mathbb{R}^p_>$. In other words, for $G(\nu)$ the subtree of G rooted in ν,

$$\forall \nu' \in G(\nu), f(X'_\nu) \cap UB_\nu + \mathbb{R}^p_> = \varnothing \tag{4.4}$$

The separation problem has been presented by Sourd and Spanjaard (2008) as that of finding a *separating function* $h : \mathbb{R}^p \to \mathbb{R}$ separating the feasible set as bounded by the upper bound set from the search region, in the sense that

$$\begin{cases} h(y) \geqslant 0 & \forall y \in f(X_\nu) \\ h(y) < 0 & \forall y \in SR \end{cases} \tag{4.5}$$

A separating function is in general non-convex, reflecting the complexity of an hypersurface separating $f(X_\nu)$ from the current incumbent set may, when such a surface exists. In practice, the separating function will be computed piece by piece. Each of these pieces defines a cone - which in extreme cases can be a half-space, such that there can be no feasible solutions within that cone. One way to ensure this condition is to generate these cones together with upper bound points. If these upper bound points are supported by hyperplanes, the pieces will be hyperplanes. Thus the following is a separating function (Sourd and Spanjaard (2008)):

$$h^\nu_\Lambda(y) = \min_{\lambda \in \Lambda}(\max_{x \in X_\nu} \langle \lambda f(x) \rangle - \langle \lambda y \rangle) \tag{4.6}$$

where $\Lambda \subseteq \mathbb{R}^p$ and $\langle \lambda, y \rangle = \sum_{j=1}^p \lambda_j y_j$. The image of a supported solution is an upper bound point, so any $y \in \mathbb{R}^p$ such that $h^\nu_\Lambda(y) < 0$ is above the hyperplane supporting the optimum of $\max_{x \in X_\nu} \langle \lambda f(x) \rangle$ for some $\lambda \in \Lambda$. In other words it is unfeasible at node ν. Note that it would also be the case for supported solutions of the linear programming relaxation of the problem under consideration at node ν. Non-supported upper bound points still provide a set of cones that can be used for separation, using the following separating function:

$$h^\nu(y) = \begin{cases} 1 & \text{if } \exists y' \in UB, y' \preceq y \\ -1 & \text{otherwise} \end{cases} \tag{4.7}$$

The reasoning is even clearer in this case. Given that no solution down in the subtree rooted at ν can dominate an upper bound point y' computed at ν, any point that would dominate y' is to be considered *unattainable at* ν, in the sense that there exists no solution $x \in X_\nu$ such that $y' \preceq f(x)$, or in other words, such that $y' \in \{f(x)\} \oplus \mathbb{R}^p_\geqslant$. In any case, to state that a point y in unattainable at ν, all we need is one piece of the separating function, either a λ such that $\max_{x \in X_\nu} \langle \lambda f(x) \rangle \leqslant \langle \lambda y \rangle$, or $\max_{x \in X^{\mathbb{R}}_\nu} \langle \lambda f(x) \rangle \leqslant \langle \lambda y \rangle$, or one upperbound solution y' such that $y \succeq y'$, where $X^{\mathbb{R}}_\nu$ is the feasible set of the linear programming relaxation the problem associated with node ν.

The approach to fathoming by bound that is exemplified by both Sourd and Spanjaard (2008) and Parragh and Tricoire (2019) is the following: if a separation function is found between SR and UB, the node is fathomed. The emptiness of $SR \cap f(X_\nu)$ is guaranteed by the fact that $f(X_\nu) \subseteq (UB - \mathbb{R}^p_\geqslant)$. If no separating function is found, branching must occur. Parragh and Tricoire (2019) note that even when separation is not possible, the hyperplanes generated while searching for a separating function provide information restricting the subspace of the objective space where non-dominated points may lie. This leads, in their line of work, to *objective space branching*.

Our own approach also aims at using information gained by generating segments of a separat-

ing function, but it differs from the other approaches in how it defines fathoming. We rely on the notion of *search zone elimination* that aims at providing two advantages. First, search zone elimination is valid for a whole subtree, and thus any child node will inherit only the zones which were not found to be unattainable at its parent node. Second, the set of upper bound points necessary to show that some zone is unattainable *needs not constitute* a strong upper bound set.

Given a node ν, consider Z_ν the set of zones to be investigated at node ν. We call these zones *active* at node ν, and will see that oftentimes, $SR_\nu := \bigcup_{z \in Z_\nu} (\{z\} \oplus \mathbb{R}^p_>)$ is a lot smaller than SR. As hinted at earlier, in practice, we do not compute the whole separating function at every node. Rather, we generate *pieces* of the separating function, and for each piece, we try to *eliminate as many search zones as possible*. We do this until either separation is achieved, or the procedure for generating pieces of the separating function terminates.

For the purpose of our implementation, let us define the function $h(c, y) : \mathcal{P}(\mathbb{R}^p) \times \mathbb{R}^p$. Where $c \subseteq \mathbb{R}^p$ is a piece of the separating function that is a cone: either the upper-right quadrant of a non-supported upper-bound solution, or the halfspace above the supporting hyperplane of an upper bound solution.

$$h(C, y) = \begin{cases} 1 & \text{if } y \in C \\ -1 & \text{otherwise} \end{cases}$$

Algorithm 27 describes the process of generating cone-like pieces of the separating function and using them to prove that $SR \cup X_\nu = \varnothing$.

Algorithm 27: Separation Algorithm (**Separate**)

input : ν a node, C_ν a set of cones generated as node ν

output: 1 if the separation succeeds, 0 otherwise

```
Z_ν ← Z_ν* where ν* is the father node of ν in G
for c ∈ C do
    for z ∈ Z_ν do
        if h(c, z) = 1 then
            Z_ν ← Z_ν\z
        if Z_ν = ∅ then
            return 1
return 0
```

Note that even when we fail to separate X_ν and SR, we may still be able to delete some zones from Z_ν. This information is useful, because for any ν' such that $(\nu, \nu') \in E$, $X_{\nu'} \subseteq X_\nu$. So, if for some $z \in Z_\nu$, $\{z\} \oplus \mathbb{R}^p_> \cap X_\nu = \varnothing$, then $\{z\} \oplus \mathbb{R}^p_> \cap X_{\nu'} = \varnothing$. Thus there is no point in further investigating, in $G(\nu)$, a zone that has been proven unattainable at node ν. The zones which remain active after the procedure has been applied are passed on to the children nodes ν' of ν.

4.4.3.4 Application to the resolution of biobjective REF

The enumeration of all decouplings described in Section C.4.4.1 will be performed as an arborescent search, in which leaf-nodes are associated with feasible sets X^σ for $\sigma \in S^{T_C}$. First, we compute, as an incumbent set, the non-dominated set of one *restrict-split* variant, that which maximizes the heuristic described in Section C.3.4. Of course, we will not be solving this variant again during the arborescent search. We also pre-compute, as in Section C.4.4.2, the local restriction which we will use to initialize each leaf-node problem. To use our BB approach, the following assertion is needed:

Observation 4.3. *Proposition C.5 implies that the decoupling branching scheme is exhaustive.*

We explore this arborescence with a depth first strategy. At each intermediary node, we try to eliminate active search zones at this node, separating them with the supporting hyperplanes of solutions to weighted sum scalarizations of the linear programming relaxation of the node problem. The weight vectors orienting these hyperplanes are generated according to the dichotomic method for the generation of supported solutions. Because we need to limit the number of supported solution we compute at each node, we defined an ε-dominance threshold. When two consecutively generated supported solutions ε-dominate one another for a value of ε smaller than this threshold, we stop. We tuned the value of the threshold manually so as to maximize the ratio of the number of zone eliminations over the number of generated supported points.

Note that we do not update the search regions when new feasible solutions are found in leaf nodes, because this process proved too costly in preliminary experiments. The overall procedure is described by Algorithm 28, where $\sigma_\nu \in S^{T_C}$ denotes the assignment of complex tasks to sites which is associated with leaf node ν, and $DPpreComp$ denotes the sub-procedure associated with lines 6 to 9 of Algorithm 26, where a *restrict-split* variant is solved using DP initialized with the precomputed decision subsequences associated with local tasks.

In the following experiment, we compare the time needed to solve REF using the e-constraint method, enumerating all *restrict-split* variants with precomputation of the local subproblems as in C.4.4.2, with the running time of the decoupling branching approach. This experiment is limited to the biobjective case, so as to take advantage of the dichotomic scheme for generating pieces of the separating hypersurface. Instances are generated in the same way as for previous experiments with REF. In Table 4.15, $T.\ Enum.\ BB$ denotes the decoupling branch and bound approach, with $\#Saved$ denoting the sum of the number of leaf nodes in the subtrees which were avoided thanks to bounding.

We can observe that for all tested series of parameters except the last one, we found the approach with bounding to perform slightly better than the enumeration of restrict-split variants with partially precomputed dynamic programming. In cases where the number of restrict-split variants to be solved was too high to allow the enumerative approach to beat the e-constraint method, using BB tricks could not reverse the tide, and e-constraint remains superior. Because we did not attempt to enrich the search region with feasible solutions obtained at leaf-nodes, we were limited

Algorithm 28: Decoupling branch and bound

input : L an incumbent set, $Q_L = (Q_L^s \mid s \in S)$ the precomputed last steps of each local tasks subproblem

output: $\tilde{\mathcal{E}}(X, f)$

$H \leftarrow v_0$ /*H is list nodes to be explored, v_0 the node associated with feasible set X of the original problem */

$SR \leftarrow InitializeSR(L)$ /*The search region is initiliazed using the incumbent set L */

$Out = L$ /*The set of results is initialized with the solution of the already computed *restrict-split* variant */

while $H \neq \varnothing$ **do**

  $v \leftarrow head(H)$

  if $ActiveZones(v) \neq \varnothing$ **then**

    if $Children(v) \neq \varnothing$ **then**

      /*This is an intermediary node */

      if $Separate(v, SR, L, S) = 0$ **then**

        for $v' \in Children(v)$ **do**

          $ActivesZones(v') = ActiveZones(v)$

        $H \leftarrow H \cup Children(v)$

    else

      /*This is a leaf node */

      if *This decoupling doesn't define the* restrict-split *solved as* L **then**

        $Out^v = DPpreComp(Q_L, \sigma_v)$

        $Out \leftarrow \mathcal{E}(Out \cup Out^v)$

  $H \leftarrow H \backslash \{v\}$

return Out

$\lvert M(s)\rvert$	$\lvert T(s)\rvert$	$\lvert T_C\rvert$	$\lvert S\rvert$	$\lvert N\rvert$	$\lvert S^{T_C}\rvert$	$\lvert ND\rvert$		$T.e-const$ *NoDec*		*T.Enum. DP.PreC Time*		*T.Enum. BB* #*Saved*		*T.Enum. BB* Time	
4	3	3	4	96	64	40.10	[12.11]	2.50	[0.91]	1.41	[0.18]	7.20	[10.73]	1.13	[0.17]
4	3	4	3	84	81	35.85	[9.80]	1.97	[0.68]	1.72	[0.13]	28.35	[30.23]	1.19	[0.24]
3	3	4	4	84	256	40.90	[16.70]	2.02	[1.20]	3.76	[0.25]	56.11	[74.31]	2.99	[0.73]
3	2	4	5	90	625	38.70	[11.85]	1.19	[0.74]	5.36	[0.56]	152.32	[218.91]	4.41	[1.46]
4	4	5	4	144	1024	78.05	[18.56]	5.53	[1.73]	78.89	[10.09]	272.80	[326.90]	72.67	[17.74]
4	4	4	5	160	625	105.75	[36.21]	8.62	[3.13]	59.56	[8.85]	101.25	[179.18]	60.81	[16.98]

Table 4.15: Computation time (in seconds) for the decoupling branch and bound approach to solving biobjective REF, as compared with the e-const approach and the enumeration of all *restrict-split* variants. Average values for 20 trials.

in our abilities to fathom nodes by the initial incumbent, and because of the experiment performed in Section 4.4.3.1, we have reasons to believe that we, in fact, only managed to eliminate a small portion of dominated leaves. Further work is thus required, both beyond the biobjective case, and thus using other upper bound notions, and to integrate quicker or more selective update of the search region when feasible solutions are obtained.

4.5 Conclusions and perspectives

In this chapter, we have explored a wide array of methods aimed at efficiently solving a discrete, multiobjective complex systems problem we called REF. Starting from the definition of a dynamic programming algorithm to solve REF, we observed that this approach suffered from an increase in the number of sites, corresponding to additional criteria introducing incomparabilities between states of the DP process. We proposed to tackle this issue in two ways.

First "bottom-up", trying to eliminate as much partial solutions as possible while relying only on local information. We used filtering using lower bound computed from the local restriction of the problem, and a kickstarted method in which we obtained the first step of the DP process associated with a complex task by aggregating results of subproblems which were solved using only local information.

Second, we proposed a "top-down" approach, the basic version of which is an exploration of all possible assignments of complex tasks to sites, which are sufficient to decouple an instance of REF. The number of instances to solve in this approach obviously grows quickly with the number of complex tasks and of sites. To compensate for this, we used DP, which efficiently solves instances with $p + 1$ criteria, and we precomputed the final stage of the local part of the problem, used to solve as many very small $p + 1$ criteria DP problems as there are decouplings.

In both cases, we precompute the local part of REF, and we pass it on the a second phase of the computation. The duality between the two approaches is clear: the number of criteria, and thus of tests between incomparable DP states in the bottom-up approach is traded for the number of instances to solve in top-down approach. Our experiments revealed the best top-down approach to perform significantly better than the best bottom-up approach.

We attempted to improve the top-down approach further by presenting the enumeration of decouplings in the form of a branch and bound search, where branching is done on the decision of assigning a complex task to a site. We obtained encouraging but contrasted and limited results in the biobjective case, which invite further research.

General conclusions and perspectives

This work has explored the multiobjective optimization of complex systems. We have broken down the subject into three main topics. First the combination of solutions obtained from the resolution of independent subproblems, then the use of decomposition to provide lower and upper bounds to the non-dominated set of a complex system MO problem, and finally we have developed applications of decompositions for the resolution of a particular complex system optimization problem, named REF.

In Chapter 2, we achieve significant performance improvements in combining and filtering sets of solutions to optimization problems. This operation is a cornerstone of methods we developed further on. We put forth two main approaches to solving this problem. The first attempted to avoid the sorting of the whole set sum, and deduce, from the sorting of individual sets, a dominance preserving order over combinations of solutions, which allows for faster dominance filtering of the set of combinations. Another approach, which did not perform as well as the previous one, involved grouping points in the sets to be combined into *boxes*. Although we have considered multiple algorithms to produce these boxes - some of which allowed for good results in applying *box-based dominance relations*, there is room for improvement in this matter. We have managed to provide a box-building algorithm, the parameters of which had to be tuned so as to provide good overall performance, albeit within to confines of a particular number of criteria. Further work should investigate whether such a box-based method could be devised without parameters, or with parameters, the values of which would remain relevant across any number of criteria.

Decomposable relaxations and restrictions of the original problem, presented in Chapter 3, proved to provide a very good approximation of the set of non-dominated solutions of the generic coupled problem. We also showed that the speed up of the computation of the bounds which is allowed by decomposition scales well with both the size of the instances and the number of criteria. This strand of research should be pursued by putting forth more refined notions of lower and upper bounds, that could be adapted to problems in which the coupling between subsystems is more complex than the one we assumed. These would include using partially relaxed or mixed constraints such as surrogate relaxations, or supported solutions which could be computed using existing methods in single objective Lagrangean relaxations. We left untouched the subject of multiobjective Lagrangean duality, because it has been developed mostly in the continuous case, which seemed more foreign to our application cases that methods we could readily develop. However, intersections between this line of inquiry and our own work are to be pursued.

Chapter 4 provided several approaches to integrating decomposable optimization and dynamic programming in a "bottom-up" fashion, both breaking down the DP process into several independent subsequence, and to quickly obtain lower and upper bounds that allow the elimination of partial solutions in large number. For $p \geqslant 3$, the relevance of *DP* was supported by the current limitations of generic algorithms. Although in most cases we achieved competitiveness for $p = 2$ as well, we observed that as the number of subsystems, and thus the size of DP states, increased, only approaches which used DP for fully decoupled problems could beat the e-constraint method. This motivated so called "Top-down" approaches, where we enumerated fully decoupled variants of the original problem so as to exhaustively cover its feasible set. We initiated a strand of work combining this enumeration of decouplings with multi-objective branch & bound, which gave promising but limited results in the biobjective case, and which we hope to pursue in future research.

Bibliography

Aneja, Y. P. and Nair, K. P. (1979). Bicriteria transportation problem. *Management Science*, 25(1):73–78.

Bazgan, C., Hugot, H., and Vanderpooten, D. (2009). Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279.

Bazgan, C., Jamain, F., and Vanderpooten, D. (2013). On the number of non-dominated points of a multicriteria optimization problem. *Discrete Applied Mathematics*, 161(18):2841–2850.

Bazgan, C., Jamain, F., and Vanderpooten, D. (2015). Approximate Pareto sets of minimal size for multi-objective optimization problems. *Operations Research Letters*, 43(1):1–6.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716.

Bentley, J. L., Clarkson, K. L., and Levine, D. B. (1993). Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica*, 9(2):168–183.

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA.

Boland, N., Charkhgard, H., and Savelsbergh, M. (2017). A new method for optimizing a linear function over the efficient set of a multiobjective integer program. *European journal of operational research*, 260(3):904–919.

Borges, P. C. and Hansen, M. P. (2002). A study of global convexity for a multiple objective travelling salesman problem. In *Essays and surveys in metaheuristics*, pages 129–150. Springer.

Changkong, Y. and Haimes, Y. (1983). Multiobjective decision making. amsterdam: Ed.

Chen, W.-M., Hwang, H.-K., and Tsai, T.-H. (2012). Maxima-finding algorithms for multidimensional samples: A two-phase approach. *Computational Geometry*, 45(1-2):33–53.

Colson, B., Marcotte, P., and Savard, G. (2007). An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256.

Cornu, M. (2017). *Local Search, data structures and Monte Carlo Search for Multi-Objective Combinatorial Optimization Problems*. PhD thesis, PSL Research University.

Delort, C. and Spanjaard, O. (2013). A hybrid dynamic programming approach to the biobjective binary knapsack problem. *Journal of Experimental Algorithmics (JEA)*, 18:1–1.

Dietz, T., Klamroth, K., Kraus, K., Ruzika, S., Schäfer, L. E., Schulze, B., Stiglmayr, M., and Wiecek, M. M. (2020). Introducing multiobjective complex systems. *European Journal of Operational Research*, 280(2):581–596.

Ehrgott, M. (2005). *Multicriteria Optimization*. Springer, Berlin-Heidelberg,, 2nd edition edition.

Ehrgott, M. and Gandibleux, X. (2001). Bounds and bound sets for biobjective combinatorial optimization problems. In *Multiple Criteria Decision Making in the New Millennium*, pages 241–253. Springer.

Escoffier, B. and Spanjaard, O. (2005). *Programmation dynamique*, pages 95–123. Lavoisier, Paris,.

Eswaran, P., Ravindran, A., and Moskowitz, H. (1989). Algorithms for nonlinear integer bicriterion problems. *Journal of Optimization Theory and Applications*, 63(2):261–279.

Figueira, J. R., Paquete, L., Simões, M., and Vanderpooten, D. (2013). Algorithmic improvements on dynamic programming for the bi-objective $\{0, 1\}$ knapsack problem. *Computational Optimization and Applications*, 56(1):97–111.

Fisher, M. L. (1981). The lagrangian relaxation method for solving integer programming problems. *Management science*, 27(1):1–18.

Gardenghi, M. (2009). *Multiobjective Optimization for Complex Systems*. PhD thesis, Clemson University.

Gardenghi, M., Gómez, T., Miguel, F., and Wiecek, M. M. (2011). Algebra of efficient sets for multiobjective complex systems. *Journal of Optimization Theory and Applications*, 149(2):385–410.

Geoffrion, A. M. (2010). Lagrangian relaxation for integer programming. In *50 Years of Integer Programming 1958-2008*, pages 243–281. Springer.

Haimes, Y. Y., Lasdon, L., and Wismer, D. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, 1(3):296–297.

Ishibuchi, H., Klamroth, K., Mostaghim, S., Naujoks, B., Poles, S., Purshouse, R., Rudolph, G., Ruzika, Stafan & Sayin, S., Wiecek, M. M., and Yao, X. (2015). Multiobjective optimization for interwoven systems. In *Report from Dagstuhl Seminar 15031: Understanding Complexity in Multiobjective Optimization*.

Jaszkiewicz, A. and Lust, T. (2018). Nd-tree-based update: a fast algorithm for the dynamic nondominance problem. *IEEE Transactions on Evolutionary Computation*, 22(5):778–791.

Kaddani, S., Vanderpooten, D., Vanpeperstraete, J.-M., and Aissi, H. (2017). Weighted sum model with partial preference information: application to multi-objective optimization. *European Journal of Operational Research*, 260(2):665–679.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer-Verlag.

Kerbérénès, A., Vanderpooten, D., and Vanpeperstraete, J.-M. (2021a). Bound sets for multiobjective optimization of complex systems. *Submitted*.

Kerbérénès, A., Vanderpooten, D., and Vanpeperstraete, J.-M. (2021b). Computing efficiently the non-dominated subset of the minkowski set sum. *Submitted*.

Kirlik, G. and Sayın, S. (2014). A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3):479–488.

Klamroth, K., Lacour, R., and Vanderpooten, D. (2015). On the representation of the search region in multi-objective optimization. *European Journal of Operations Research*, 245(3):767–778.

Klamroth, K. and Wiecek, M. M. (2000). Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics (NRL)*, 47(1):57–76.

Kung, H.-T., Luccio, F., and Preparata, F. P. (1975). On finding the maxima of a set of vectors. *Journal of the ACM (JACM)*, 22(4):469–476.

Laumanns, M., Thiele, L., and Zitzler, E. (2006). An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–942.

Lemaréchal, C. (2001). Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer.

Liu, B., Bissuel, C., Courtot, F., Gicquel, C., and Quadri, D. (2021). A hierarchical decomposition approach for the optimal design of a district cooling system. In *ICORES*, pages 317–328.

Lokman, B. and Köksalan, M. (2013). Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, 57(2):347–365.

Martello, S. and Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, 28(1):59–70.

Migdalas, A., Pardalos, P. M., and Värbrand, P. (2013). *Multilevel optimization: algorithms and applications*, volume 20. Springer Science & Business Media.

Nakayama, H. (1984). Geometric consideration of duality in vector optimization. *Journal of Optimization Theory and Applications*, 44(4):625–655.

Nakayama, H. (1985). Duality theory in vector optimization: an overview. In *Decision making with multiple objectives*, pages 109–125. Springer.

Nemhauser, G. L. and Wolsey, L. A. (1998). *Integer and Combinatorial Optimization*. Wiley-Interscience, New York,.

Parragh, S. N. and Tricoire, F. (2019). Branch-and-bound for bi-objective integer programming. *INFORMS Journal on Computing*, 31(4):805–822.

Przybylski, A. and Gandibleux, X. (2017). Multi-objective branch and bound. *European Journal of Operational Research*, 260(3):856–872.

Sayın, S. and Kouvelis, P. (2005). The multiobjective discrete optimization problem: A weighted min-max two-stage optimization approach and a bicriteria algorithm. *Management Science*, 51(10):1572–1581.

Serafini, P. (1987). Some considerations about computational complexity for multi objective combinatorial problems. In *Recent advances and historical development of vector optimization*, pages 222–232. Springer.

Sourd, F. and Spanjaard, O. (2008). A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3):472–484.

Stiglmayr, M., Figueira, J. R., and Klamroth, K. (2014). On the multicriteria allocation problem. *Annals of Operations Research*, 222(1):535–549.

Tamby, S. and Vanderpooten, D. (2021). Enumeration of the nondominated set of multiobjective discrete optimization problems. *INFORMS Journal on Computing*, 33(1):72–85.

TenHuisen, M. L. and Wiecek, M. M. (1994). Vector optimization and generalized lagrangian duality. *Annals of Operations Research*, 51(1):15–32.

Ulungu, E. L. and Teghem, J. (1995). The two phases method: An effcient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision*, 20(2):149–165.

Vassilvitskii, S. and Yannakakis, M. (2005). Efficiently computing succinct trade-off curves. *Theoretical Computer Science*, 348(2-3):334–356.

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. *Journal of Global Optimization*, 12(2):139–155.

Yildiz, B., Boland, N., and Savelsbergh, M. (2018). Decomposition branching for mixed integer programming. Technical report, Tech. rep. 2018. url: http://www. optimizationonline. org/DB_HTML/2018/08 ....

Appendices

Appendix A

The double bound as a solution concept in MOCO

A.1 Definitions

In some operational situations, knowledge of all non-dominated points may not be the most useful information to a decision maker, especially in contexts that require quick decision making. In such contexts, the decision maker may be interested in being presented with bounds to the non-dominated set, so that she can use this information to trigger another multicriteria decision method.

For example, the information provided by either a strong or weak upper bound, and by either a weak or strong lower bound, may help the decision maker select an aspiration point to be approached by means of a scalarized method, knowing which level of performance are atteinable. In other contexts, when both a feasible, lower bound solution and an upper bound point are provided, the latter may be used as a quality estimate of the former. The decision maker may decide to settle for the feasible point if the closest upper bound point that dominates is sufficiently close to it.

Such a *double bound* materializes the uncertainty of the decision maker regarding the location of the non-dominated set, in the form of an area or hypervolume. One way to quantify this uncertainty, given X the feasible set of the problem and f the vector valued objective function, is to produce an estimate of the probability that a point drawn randomly and bounded by an upper bound and a lower bound on $f(X)$ will fall "between" the LB set and the UB set we computed.

What we mean by "between" needs to be made precise and depends on whether the UB and LB sets are weak or strong bounds to $\mathcal{N}(f(X))$.

Definition A.1. *If LB is a* weak *lower bound (wLB) to* $\mathcal{N}(f(X))$*, then* $y \in \mathbb{R}^p$ *is* above *LB if*

$$y \notin LB \ominus \mathbb{R}^p_{\geq}$$

If LB is a strong *lower bound (sLB), then* $y \in \mathbb{R}^p$ *is* above *LB if*

$$y \in LB \oplus \mathbb{R}^p_{\geq}$$

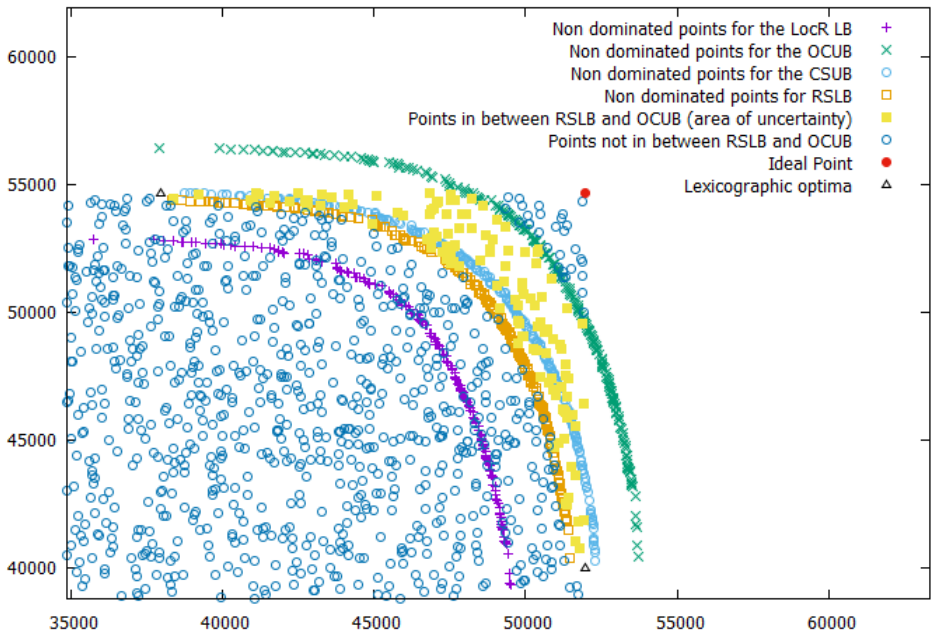


Figure A.1: An example of a double bound and a sample of points drawn randomly to measure area of uncertainty after computing the RS wLB and the LRUB

Definition A.2. *If UB is a* weak *upper bound (wUB), then $y \in \mathbb{R}^p$ is* below *UB if*

$$y \notin UB \oplus \mathbb{R}^p_{\geq}$$

If UB is a strong *upper bound (sUB), then $y \in \mathbb{R}^p$ is* below *UB if*

$$y \in UB \ominus \mathbb{R}^p_{\geq}$$

Definition A.3. *Given $\mathcal{N}(f(X))$, UB either a strong or weak upper bound set to $\mathcal{N}(f(X))$ and LB either a strong or weak lower bound on $\mathcal{N}(f(X))$, $y \in \mathbb{R}^p$ falls* between *LB and UB if it is above LB and below UB.*

Within this section, we do not present any strong lower bound concept for the GCP. However, for application problems presented in later sections of this work, we will.

Figure A.1 features a graphical representation of the *local restriction* (LocR) and *restrict-split* (RS) weak lower bounds in purple and orange respectively, and the *copy-split* (CS) and *local omission of coupled variables* (LR) upper bounds in light blue and green respectively. Secondly, we plot the randomly sampled points used to quantify the decision maker's uncertainty, conditioned on the computation of the RS and LR bounds only. Dark blue points are those which are either dominated by a point in the RS wLB, or dominate a point in the LR UB, while yellow points are those which fall in the area of uncertainty. Computing RS and CS yields much lower uncertainty, but plotting the resulting area of uncertainty would produce a less visible an thus unclear example. Next, we give further details on the computation which provides this result.

p	n	$\|S\|$	$\|K\|$	DV	LocR/CS		LocR/LR		LocR+CS		LocR+LR		Idl/(LcR+CS)		Idl/(LcR+LR)	
2	100	2	1	10	1.571	[0.65]	2.342	[0.83]	3.118	[1.11]	2.707	[0.98]	1.874	[0.75]	2.152	[0.82]
2	120	2	1	10	1.59	[0.42]	2.33	[0.49]	4.167	[0.92]	3.581	[1.04]	1.509	[0.59]	1.84	[0.88]
2	140	2	1	10	2.135	[0.76]	2.961	[0.67]	6.812	[1.64]	6.172	[1.55]	1.085	[0.38]	1.214	[0.48]
2	100	3	1	10	1.733	[0.78]	2.347	[0.64]	1.894	[0.58]	1.685	[0.53]	2.996	[1.03]	3.431	[1.29]
2	100	4	1	10	1.454	[0.63]	2.007	[0.62]	1.172	[0.46]	1.056	[0.42]	5.352	[1.83]	6.019	[2.12]
2	100	5	1	10	1.21	[0.54]	1.76	[0.61]	0.904	[0.3]	0.776	[0.28]	6.611	[2]	7.854	[2.63]
2	100	2	2	10	2.987	[0.75]	3.599	[1.05]	2.597	[0.72]	1.945	[0.47]	2.469	[0.76]	3.232	[0.85]
2	100	2	3	10	3.286	[0.89]	3.948	[1.13]	2.4	[0.77]	1.501	[0.54]	2.789	[1.62]	4.705	[3.62]
2	100	2	4	10	4.08	[1.39]	4.604	[1.5]	2.285	[0.67]	1.288	[0.51]	3.298	[1.38]	6.224	[2.99]
2	100	2	1	20	3.511	[0.89]	4.484	[1.14]	3.204	[0.81]	2.352	[0.64]	2.161	[0.89]	2.959	[1.24]
2	100	2	1	30	4.106	[1.02]	4.874	[1.37]	2.128	[0.61]	1.36	[0.44]	3.059	[0.96]	4.933	[2.07]
2	100	2	1	40	4.791	[1.51]	5.452	[1.73]	2.2	[0.56]	1.222	[0.43]	3.46	[1.11]	6.598	[2.81]
2	40	2	1	10	1.108	[0.49]	1.731	[0.55]	0.331	[0.1]	0.287	[0.09]	8.83	[3.59]	10.55	[5.18]
3	40	2	1	10	2.433	[1.57]	3.97	[1.73]	3.666	[2.66]	3.224	[2.31]	1.603	[1.12]	1.852	[1.32]
4	40	2	1	10	4.384	[2.61]	7.423	[1.89]	119.2	[143.3]	68.28	[72.79]	0.159	[0.18]	0.205	[0.21]

Table A.1: Quality of the double bound, as hypervolume difference and as a posteriori ε-dominance. Average values for 20 trials.

The correctness of our measurement requires that the lower bound of the hyperrectangle in which we sample points be a lower bound point to $f(X)$, and accordingly that the upper bound of the same hyperrectangle be an upper bound on $f(X)$. Assuming that for each $j \in \{1, ..., p\}$, f_j is linear with positive coefficients, we can take $\mathbf{0}$ as a lower bound. As for the upper bound on the hyperrectangle, we propose two alternatives. They differ with regards to the sort of *a priori* information about $f(X)$ that they require from the decision maker. In the first case, under the same hypotheses about $f(X)$, the only required information are that coefficients p^i_j be known. Then we set u^{max} such that for each $j \in \{1, ..., p\}$, $u^{max}_j = \sum_{i=1}^{n} p^i_j$. It is easily seen that this is the maximum of $f^j(x)$ for $x \in \{0, 1\}^n$, i.e. if the optimization of f^j was unconstrained, and thus $y \preceq u^{max}$ for any $y \in f(X)$. In the second case, we use the *ideal point* of $f(X)$ as upper bound.

The use of the *ideal point* as an upper bound on random points is well suited to the case where our measurement is meant to quantify the decision maker's uncertainty, but it is not well suited when it is meant to quantify the quality of bound concepts that do not involve the ideal point itself. Indeed, in some cases, the ideal point may be dominated by some point in the upper bound set, and thus constitute by itself a tighter upper bound, the knowledge of which reduces uncertainty. In much the same way, the lexicographic optima obtained by computing the ideal point constitute lower bound points which may dominate points in the (weak) lower bound under scrutiny. In a context when uncertainty is to be quantified, one should add these points to the bound sets, and add the time spent computing the ideal point to the time spent computing the double bound.

A.2 Experimental Results

In practice, we draw 10000 points uniformly randomly in a hypersquare box of lower bound $\mathbf{0}$ and upper bound u^{max} or ideal point. In this experiment, LB is always a weak lower bound set, so for each random point r drawn, we call the draw a success if there is no $y \in LB$ such that $r \preceq y$, and if there is a $y' \in UB$ such that $r \preceq y'$. Our metric is then simply the proportion of successful draws

p	n	$\|S\|$	$\|K\|$	DV	RS/CS		RS/LR		RS+CS		RS+LR		Idl/(RS+CS)		Idl/(RS+LR)	
2	100	2	1	10	0.436	[0.42]	1.192	[0.54]	3.302	[1.3]	2.891	[1.13]	1.818	[0.81]	2.054	[0.85]
2	120	2	1	10	0.693	[0.33]	1.444	[0.47]	4.402	[1.02]	3.816	[1.05]	1.433	[0.57]	1.69	[0.72]
2	140	2	1	10	0.691	[0.40]	1.468	[0.27]	7.097	[1.83]	6.457	[1.69]	1.045	[0.36]	1.159	[0.44]
2	100	3	1	10	0.89	[0.46]	1.538	[0.42]	1.899	[0.58]	1.691	[0.52]	2.998	[1.04]	3.402	[1.23]
2	100	4	1	10	0.69	[0.34]	1.254	[0.4]	1.213	[0.44]	1.098	[0.4]	5.145	[1.94]	5.72	[2.11]
2	100	5	1	10	0.588	[0.28]	1.145	[0.28]	0.951	[0.32]	0.824	[0.29]	6.256	[1.82]	7.319	[2.29]
2	100	2	2	10	1.214	[0.48]	1.898	[0.62]	2.869	[0.85]	2.216	[0.54]	2.265	[0.79]	2.853	[0.84]
2	100	2	3	10	1.534	[0.46]	2.142	[0.67]	2.772	[1.04]	1.873	[0.76]	2.454	[1.31]	3.69	[2.17]
2	100	2	4	10	2.214	[0.71]	2.787	[0.8]	2.758	[0.81]	1.761	[0.55]	2.78	[1.28]	4.365	[1.94]
2	100	2	1	20	1.624	[0.81]	2.579	[0.82]	3.51	[0.91]	2.658	[0.66]	1.948	[0.7]	2.549	[0.89]
2	100	2	1	30	2.496	[0.91]	3.227	[1.01]	2.613	[0.76]	1.845	[0.51]	2.491	[0.78]	3.438	[0.81]
2	100	2	1	40	2.908	[1.06]	3.64	[1.28]	2.878	[0.83]	1.9	[0.49]	2.705	[0.96]	4.024	[1.29]
2	40	2	1	10	0.204	[0.36]	0.862	[0.29]	0.363	[0.11]	0.319	[0.1]	8.296	[3.86]	9.679	[5.48]
3	40	2	1	10	0.612	[0.82]	2.13	[1.2]	3.858	[2.86]	3.416	[2.48]	1.479	[0.98]	1.639	[1.02]
4	40	2	1	10	1.101	[1.53]	4.097	[1.54]	149.1	[188]	98.22	[113]	0.153	[0.18]	0.173	[0.19]

Table A.2: Quality of the double bound, as hypervolume difference and as a posteriori ε-dominance. Average values for 20 trials.

expressed as percentage. In Tables A.1 and A.2, results for this metric are denoted by *LB/UB*, where *LB* stands in turn for *LocR* and *RS*, and *UB* for *CS* and *LR*. $LB + UB$ denotes the computing time of the two bounds. $Idl/(LB + UB)$ denotes the proportion of the total computing time spent computing the ideal point.

For any value of p, and any parameter set, we find that a double bound obtained using the *RS* and lexicographic optima weak lower bound, and the *CS* and ideal point upper bound yielded the smallest area of uncertainty of all four combinations of bound notions. Expectedly, the size of the area of uncertainty for any notion of double bound clearly increases with an increase in $|K|$ and in DV, while it appears to slightly increase with an increase of n, although for RS/CS, this trend it tainted with great variability.

Appendix B

Weak lower bounds obtained from correcting weak upper bound solutions

B.1 Correction of solutions to the copy-split relaxation

Recall that the copy-split relaxation of REF, which is equivalent to a variant in which for each $t \in T_C$, the collection of assignments variables $(x_{tm} \mid m \in M(S(t))$ is replaced by $((x_{tm} \mid m \in M(S(t))) \mid s \in S)$ i.e. by a collection of copies of the complex task assignments, each restricted to a subsystem. This relaxation is formulated as

$$
\begin{aligned}
\max \quad & \sum_{t \in T} \sum_{m \in M(S(t))} g^j_{tm} x_{tm} && \forall\, j \in \{1, ..., p\} \\
s.t. \quad & \sum_{m \in M(S(t))} x_{tm} \leqslant 1 && \forall\, t \in T, s \in S \\
& \sum_{t \in T(s)} \sum_{m \in M(S(t))} w_{tm} x_{tm} \leqslant b_s && \forall\, s \in S \\
& x_{tm} \in \{0, 1\} && \forall\, t \in T,\ m \in M(S(t))
\end{aligned}
$$

This relaxation, in practice, gives an upperbound which is a lot looser than the relaxation of coupled variables from local knapsack constraints, and contrary to the latter, its quality decreases as the number of subsystems increases. However we can use it to another end. Solutions to this problem may be infeasible as they violate the assignment constraints of complex tasks assignment variables, but they can be made feasible easily by just choosing one among the several assignments of a complex task. Given a solution for the relaxation, we make the choice of which task assignment to undo in the following way. Contrarily to greedy approaches where we want to add assignments, here we sort assignment, for each $j \in \{1, ..., p\}$, by *increasing* order of $\frac{g^j_{tm}}{w_{tm}}$, and then by increasing sum of their objective-wise ranks. Thus, we undo the "worst" assignments first, freeing more budget by sacrificing less profit. Until all complex tasks are assigned at most once, we consider in order each assignment (t, m). If it has been made in the solution, while the task has also be assigned to another machine m', we undo assignment (t, m). This is described by

Algorithm 29.

Algorithm 29: Greedy correction of solutions of the copy-split relaxation

input : $Asg, \mathcal{E}(X', f)$

output: $Corrected \subseteq X$

$Corrected \leftarrow \varnothing$
for $x \in \mathcal{E}(X', f)$ **do**
  while $x \notin X$ **do**
    for $(t, m) \in Asg$ **do**
      if $\sum_{m \in M(S(t))} x_{tm} \geqslant 1$ **then**
        $x \leftarrow x[x_{tm} \leftarrow 0]$
  $Corrected \leftarrow Corrected \cup \{x\}$
return $Corrected$

B.2 Correction of solutions to the relaxation of coupled variables in subsystems knapsack constraints

We recall that the relaxation of coupled variables in subsystems knapsack constraints yields the following problem, the feasible set X' of which has been proven to be a superset of the feasible set X of the original problem.

$$
\begin{aligned}
\max \quad & \sum_{t \in T} \sum_{m \in M(S(t))} g^j_{tm} x_{tm} && \forall\, j \in \{1, ..., p\} \\
s.t. \quad & \sum_{m \in M(S(t))} x_{tm} \leqslant 1 && \forall\, t \in T \\
& \sum_{t \in T_L(s)} \sum_{m \in M(S(t))} w_{tm} x_{tm} \leqslant 1 && \forall\, s \in S \\
& x_{tm} \in \{0, 1\} && \forall\, t \in T,\ m \in M(S(t))
\end{aligned}
$$

Solutions to this relaxation may be such that the original knapsack constraint, taking into account the coupled variables, are violated. To obtain a feasible solution, we therefore need to undo some of the assignments done in these solutions. Since we want to obtain a feasible solution that is as close as possible to the Pareto set, we wish to undo assignments so as to decrease objective value as little as possible. Here consider assignments (t, m) for $t \in T, m \in M(S(t))$ in the same order as for the previous correction concept, yielding sequence Asg. Until the knapsack constraint for each of the subsystems is satisfied, we consider in order each assignment, and if it is done in a subsystem for which the knapsack constraint is violated, we remove it. This procedure is described by Algorithm 30.

Algorithm 30: Greedy correction of solutions of the LR relaxation

input : $Asg, \mathcal{E}(X', f)$ the set of efficient solutions of the LR Relaxation.

$Corrected \leftarrow \varnothing$
for $x \in \mathcal{E}(X', f)$ **do**
  while $x \notin X$ **do**
    for $(t, m) \in Asg$ **do**
      Let s be the unique $s \in S(t)$ such that $m \in M(S(t))$
      if $\bar{w}_s(x) < 0$ **then**
        $x_{tm} \leftarrow 0$
  $Corrected \leftarrow Corrected \cup \{x\}$
return $Corrected$

B.3 Experimental results

We perform experiment regarding the computation of an incumbent set from corrections of solutions to the *CS* and *LR* upper bounds, a process which we have described in Sections B.1 and B.2 respectively. We compare the time required to obtain these incumbent sets from the upper bound computed either with the *e*-constraint method or the DP method, and we measure how well they approximate the efficient set of the original problem.

p	$\lvert M(s)\rvert$	$\lvert T_C\rvert$	$\lvert S\rvert$	*CorLR*						*CorCS*					
				$T.e-cst\&Dec$		$T.DP\&Dec$		$ApQ.(\varepsilon)$		$T.e-cst\&Dec$		$T.DP\&Dec$		$ApQ.(\varepsilon)$	
2	7	1	2	1.092	[0.494]	0.205	[0.052]	0.069	[0.021]	1.363	[0.589]	0.285	[0.066]	0.044	[0.016]
2	8	1	2	1.777	[0.52]	0.516	[0.156]	0.065	[0.019]	1.888	[0.584]	0.721	[0.205]	0.033	[0.011]
2	9	1	2	2.821	[1.566]	1.201	[0.331]	0.057	[0.018]	3.343	[1.548]	1.76	[0.528]	0.028	[0.008]
2	6	1	2	0.525	[0.198]	0.081	[0.02]	0.075	[0.026]	0.735	[0.265]	0.114	[0.032]	0.047	[0.023]
2	6	1	3	0.844	[0.339]	0.277	[0.098]	0.047	[0.023]	1.093	[0.402]	0.27	[0.039]	0.06	[0.019]
2	6	1	4	1.266	[0.69]	1.09	[0.535]	0.04	[0.013]	1.774	[1.143]	0.586	[0.124]	0.051	[0.014]
2	7	2	2	1.443	[0.956]	0.286	[0.073]	0.092	[0.039]	1.99	[0.985]	0.482	[0.112]	0.088	[0.035]
2	7	3	2	1.97	[1.344]	0.407	[0.152]	0.124	[0.035]	2.719	[1.177]	0.857	[0.277]	0.138	[0.054]
3	4	2	2	0.653	[0.14]	0.069	[0.04]	0.159	[0.041]	2.479	[0.809]	0.041	[0.013]	0.172	[0.057]
4	4	2	2	0.789	[0.161]	0.281	[0.194]	0.124	[0.052]	5.528	[2.433]	0.074	[0.025]	0.146	[0.053]

Table B.1: Computation of *LBs* obtained by greedily correcting solution from the *CS* and *LR* upper bounds. Average values for 20 trials.

From Table B.1 we can observe that, because the correction process is quick and linear, these incumbents are obtained in virtually the same time as the bound sets they originate from. However, this time is itself rather long, and generally less than 10 times quicker than the resolution of the whole problem, except of course when $p \geqslant 3$. Finally, we can observe that correcting solutions from the *LRUB* usually provides a tighter bound, which makes it the better of the two options in cases the *LRUB* is computed quicker than *CSUB* (i.e. for low values of $|S|$).

Appendix C

Résumé en Français

C.1 Notions fondamentales

C.1.1 Notions d'optimisation multiobjectifs

Pour un ensemble d'indices de variables $N = \{1, ..., n\}$, $X \subseteq \mathbb{R}^n$ un ensemble contraint $f : \mathbb{R}^n \to \mathbb{R}^p$ une fonction à valeur vectorielle, le problème d'optimisation multiobjectif associé avec f et X s'écrit

$$\begin{aligned} \max \quad & (f_1(x), ..., f_p(x)) \\ & x \in X \end{aligned}$$

Un point image de x par f peut être désigné par$y := (y_1, ..., y_p) = f(x)$

Definition C.1. *Pour $y, y' \in \mathbb{R}^p$, $y \succeq y'$, si y est au moins aussi bon quey' sur tous les objcetifs et meilleur un un objectif au moins.*

$$y \succeq y' \Leftrightarrow \begin{cases} \forall j \in \{1, ..., p\}, & y_j \geqslant y'_j \\ \exists k \in \{1, ..., p\}, & y_k > y'_k \end{cases}$$

Definition C.2. *Un point $y \in Y$ est non dominé s'il existe aucun $y' \in Y$ tel que $y' \succeq y$, faiblement non dominé si il n'existe aucun $y' \in Y$ tel que $y' \succ y$.*

Definition C.3. *Une solution $x \in X$ est efficace par rapport à f si et seulement si il n'existe acun $x' \in X, x'$ tel que $f(x') \succeq f(x)$, et est faiblement efficace si et seulement si il n'existe aucun $x' \in X$ tel que $f(x') \succ f(x)$.*

Ainsi, il est fréquent de considérer comme concept de solution l'énumération de l'ensemble des solutions effiaces ou l'ensemble de ponts non dominés:

$$\mathcal{E}(X, f) := \{x \in X \mid x \text{ est efficace relati } f\} \mathcal{N}(Y) := \{y \in Y \mid y \text{ is non-dominated}\}$$

Definition C.4. *Une solution efficace* $x' \in \mathcal{E}(X, f)$ *est supportée s'il existe* $\lambda = (\lambda_j \mid j \in \{1, ..., p\}) \in \mathbb{R}^p_>$ *tel que* $x' = \arg\max_{x \in X} \sum_{j=1}^{p} \lambda_j f_j(x)$. *Quand* $p = 2$, *la méthode dichotomique de Aneja and Nair (1979) énumère tous les points supportés.*

Definition C.5. *Pour* $y = (y_1, y_2, ...y_p)$ *et* $y' = (y'_1, y'_2..., y'_p) \in \mathbb{R}^p$, $y \succeq_{Lex} y'$ *si et seulement si* $y_1 > y'_1$, *ou* $y_1 = y'_1$ *et* $(y_2, ...y_p) \succeq_{Lex} (y'_2, ..., y'_p)$.

C.1.1.1 Algorithms de filtrage par dominance

L'algorithme de filtrage par dominance le plus simple parcours une liste de points Y et en compare chaque élément à ceux de l'ensemble de candidats $\tilde{Y}$, points jusqu'ici non dominés. Si aucun $y' \in \tilde{Y}$ ne domine y, il est ajouté à $\tilde{Y}$. Cet algorithme peut être amélioré en triant Y lexicographiquement (see Kung et al. (1975)). Alors, un point ne peut dominer son prédécesseur, et la moitié des tests dominance sont évités. Quand $p = 2$, un seul test est nécessaire car si y est dominé dans $\tilde{Y}$, il est dominé par son dernier élément.

Alternativement, le filtrage peut être fait par insertion dans un *KDTree* au sens de Chen et al. (2012), au sein d'une méthode en deux phases décrite par l'Algorithme 2. Un point dénommé *sieve* permet de commencer la série de tests par un test ayant de bonnes chance de réussir. Une routine de pruning permet un filtrage intermédiaire du KDTree et une amélioration des performances.

C.1.1.2 Choix d'un algorithme de filtrage par dominance

Quand $p = 2$, on utilise l'algorithme unidirectionnel décrit par Kung et al. (1975), avec filtrage en temps constant. Quand $p \geqslant 4$, on utilise la méthode en deux phases avec insertion dans un KDTree de Chen et al. (2012). Qand $p = 3$, il faudra décider au cas par cas entre unidirectionnel et insertion dans un KDTree.

C.1.2 Problèmes coupés et décomposition

C.1.2.1 Formulations décomposables

Soit $N = \{1, ..., n\}$ l'ensemble des variables décisions partionné par S. Un sous-système $s \in S$ est identifié par un sous-sytèmes de variables. Thus, $x \sim (x_1, ..., x_n)$, ou $(x^s | s \in S)$, où $x^s = (x_i \mid i \in s) = (x^s_1, ..., x^s_{n_s})$. Les interactions entre sous-systèmes sous représentées par des contraintes associant des variables de différents sous-systèmesL Soit $K \subseteq \mathcal{P}(N)$ un ensemble de couplages représentés par les ensembles d'indices de variables couplées.

Definition C.6. $s, s' \in S, s \neq s'$ *sont couplés par la contrainte* k *s'il existe* $i \in s \cap k$ *et* $j \in s' \cap k$. *Un problème d'optimisation est non couplés si et seulement si* $K = \varnothing$.

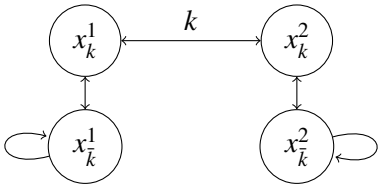


Figure C.1: Graph representation of a coupled system

Soit x^k le vecteur des variables de décision couplées par k, et x_k^s les variables de sous-système s couplées par k, i.e. $x_k^s = (x_i \mid i \in s \cap k)$. X^s dénote l'ensemble réalisable du sous-problème associé avec le sous-système $s \in S$. X^k dénote le sous-ensemble réalisé défini par la contrainte k sur le sous-vecteur de variables x^k. Ainsi, le problèle d'optimisation de système couplés est:

$$\begin{array}{lll} \max & f(x) = (f_1(x), ..., f_p(x)) & \\ s.t. & x^k \in X^k & \forall\, k \in K \\ & x^s \in X^s & \forall\, s \in S \end{array} \qquad (P)$$

avec, pour tout $j \in \{1, ..., p\}$, $f_j(x) = \sum_{s \in S} f_j^s(x^s)$. Si $|S| = m$ et $(x^s | s \in S) = (x^1, ..., x^m)$, on dénote par $x^{\bar{s}}$ les sous-vecteurs de variables n'appartenant pas à s.

Observation C.1. *Si un problème (P) est non couplés, son ensemble réalisable X est tel que*

$$X = \prod_{s \in S} X^s \qquad (C.1)$$

C.1.2.2 Optimisation découplée

Si $f_j(x)$ est additivement séparable pour tout $j \in \{1, ..., p\}$, la fonction objectif du sous-problème associé à $s \in S$ est donné par $f^s(x^s) = (f_1^s(x^s), ...f_p^s(x^s))$, avec $f_j^s : \prod_{i \in s} X_i \to \mathbb{R}$ pour $j \in \{1, ..., p\}$. Chaque sous-système est associé avec le problème $\max_{x^s \in X^s} f^s(x^s)$.

Definition C.7. *Pour $j \in \{1, ..., p\}$, $f_j : \prod_{i \in N} X_i \to \mathbb{R}^p$, et pour tout $s \in S$ $f_j^s : \prod_{i \in s} X_i \to \mathbb{R}^p$, f_j est additivement separable selon S si, pour tout $x = (x^s \mid s \in S)$ avec $x^s = (x_i \mid i \in s)$,*

$$f_j(x) = \sum_{s \in S} f_j^s(x^s)$$

Une question d'intérêt pour la généralisation multiobjectif de la décomposition est la suivante, où $\overset{\circ}{\sum}_{s \in S} Y^s$ dénote la somme de Minkowski.

$$\mathcal{E}(X, f) \overset{?}{=} \prod_{s \in S} (\mathcal{E}(X^s, f^s))$$

ou, dans l'espace des objectifs

$$\mathcal{N}(f(X)) \overset{?}{=} \overset{\circ}{\sum_{s \in S}} \mathcal{N}\,(f^s(X^s))$$

Proposition C.1. *Si* $\prod_{s\in S} X^s = X$ *et si pour tout* $j \in \{1, ..., p\}$, f_j *est séparable selon* S, *alors*

$$\mathcal{N}(f(X)) \subseteq \overset{\circ}{\sum_{s\in S}} \mathcal{N}(f^s(X^s))$$

Si $p = 1$, on a aussi $\max_{x\in X} \sum_{s\in S} f^s(x^s) = \sum_{s\in S} \max_{x^s\in X^s} f^s(x^s)$. Mais pour $p \geqslant 2$, $\sum_{s\in S} \mathcal{N}(f^s(X^s)) \notin \mathcal{N}(f(X))$. Un autre filtrage par dominance est donc nécessaire.

C.1.3 Efficacité de la décomposition

L'efficacité de la décomposition est testée en résolvant un problème générique non couplés, d'abord en utilisant un algorithme générique sur le problème entier, puis en résolvant les sous-problèmes séparément, en combinant les solutions et en filtrant le résultat par dominance. Les résultats de cette expérience suffisent à montrer que la décomposition permet une amélioration significative de la performance pour n'importe quel nombre de critères, et pour une augmentation du nombre de sous-systèmes et du nombre de variables.

C.2 Calcul efficace du sous ensemble non-dominé d'une somme d'ensembles

C.2.1 Définition du problème

Il s'agit de calculer efficacement $\mathcal{N}(\overset{\circ}{\sum}_{s\in S} Y^s)$, avec $Y^s \subset \mathbb{R}^p$ pour tout $s \in S$. Le défi principal de ce calcul est d'organiser correctement la sommation et le filtrage afin de générer le point de combinaisons dominées possible.

C.2.1.1 Filtrage intermédiaire

Un algorithme séquentiel considère les ensembles Y^s l'un après l'autre, et à chaque étape $s \in S$, génère les combinaisons entre points de Y^s et points de $Y^{\overrightarrow{s-1}}$. Plutôt que de générer la somme pour les tous les $s \in S$ puis filtrer, le résultat suivant assure qu'il est correct d'appliquer un filtrage par dominance à chaque étape.

Proposition C.2. *Pour toute famille* $(Y^s \mid s \in S)$ *avec* Y^s *fini pour chaque* $s \in S$ *et* $S = \{1, ..., n\}$, *pour tout* $s' \leqslant n$,

$$\mathcal{N}\left(\overset{\circ}{\sum_{1\leqslant s\leqslant s'}} Y^s\right) = \mathcal{N}\left(\mathcal{N}\left(\overset{\circ}{\sum_{1\leqslant s\leqslant s'-1}} Y^s\right) \oplus Y^{s'}\right)$$

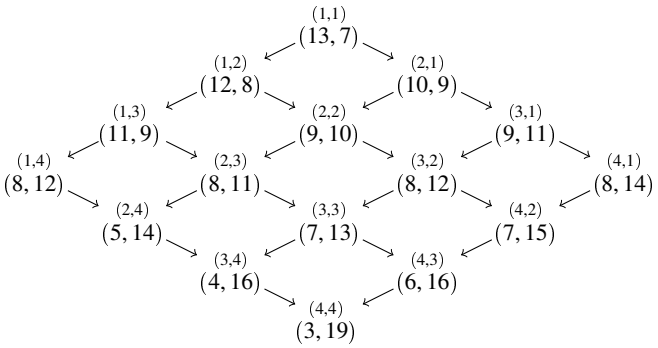


Figure C.2: Lattice représentant la relation de parenté sur $Y \oplus Z$ dans l'Exemple C.1

$T =$

	1	2	3	4
1	0	1	1	1
2	1	2	2	2
3	1	2	2	2
4	1	2	2	2

Figure C.3: Tableau représentant la relation de parenté sur $Y \oplus Z$ dans l'Exemple C.1

C.2.1.2 Résultats expérimentaux

L'expérience suivante mesure l'efficacité de l'application du filtrage intermédiaire à chaque étape de la résolution de NDMSP par un algorithme séquentiel. Le gain de temps (TG), est défini comme $100\frac{T.NA-T.IF}{T.NA}$, où $T.NA$ dénote le temps de calcul lorsqu'un seul filtrage est fait après la somme, et $T.IF$ en filtrant à chaque étape. L'écart type est rapporté entre crochets.

C.2.1.3 Un algorithme de *pooling* unidirectionnal

C.2.1.4 Définition et preuve de correction

Supposant que Y et Z sont triés lexicographiquement et leurs éléments respectifs indicés d'après leurs rangs. On définit un treillis tel que chaque $y^i + z^j \in Y \oplus Z$ a au plus deux enfatns $y^{i+1} + z^j$ et $y^i + z^{j+1}$, if $i+1 \leqslant |Y|$ et $j+1 \leqslant |Z|$ et au plus deux parents $y^{i-1} + z^j$ et $y^i + z^{j-1}$, if $i-1 > 0$ and $j+1 > 0$. Ce treillis est un sous ensemble de $\succeq_{Lex}$ sur $Y \oplus Z$. Le tableau T enrgistre, en chaque entrée (i, j), le nombre de parents de $y^i + z^j \in Y \oplus Z$ non testés pour insertion.

Example C.1. *Soit* $Y = \{(6,3),(3,5),(2,7),(1,10)\}$ *et* $Z = \{(7,4),(6,5),(5,6),(2,9)\}$*, triés lexicographiquement et indicés.* T *représente par un tableau la relation du treillis, cf. Figure C.3.*

$P \subseteq \mathcal{N}(Y \oplus Z)$ représente l'ensemble $\mathcal{N}(Y \oplus Z)$ en construction et H, trié lexicographiquement, le sous ensemble de $Y \oplus Z$ dont les entrées dans T sont à 0. L'élément au sommet de H est le prochain élément à insérer dans P de sorte à respecter un ordre lexicographique sur $Y \oplus Z$.

IF et UPool peuvent être comparés du point de vue de la complexité. Supposons que $|Y| = m$, $|Z| = n$, with $m \geqslant n$. IF et UPool diffèrent dans la phase de tri préalable au filtrage par algorithme unidirectionnel. IF nécessite de trier $Y \oplus Z$, en temps $O(mn \log mn) = O(mn \log m)$. UPool nécessite de trier Y et Z indépendamment, en temps $O(m \log m + n \log n) = O(m \log m)$, puis d'insérer les mn elements dans H. Il est possible de prouver que H est toujours de taille inférieure à n. L'opération est donc réalisée en temps $O(mn \log n)$. Comme dans le cas biobjectif, le tri représente la majorité de temps de calcul (l'insertion étant en temps constant) UPool a de bonnes chances d'être plus performant que IF dans ce cas, où UPool a une complexité de $O(mn \log n)$ et IF de $O(mn \log m)$.

C.2.1.5 Résultats expérimentaux

Pour $p = 2$ on trouve expérimentalement que UPool offre des performances jusqu'à $81,65$ % supérieures à IF, augmentant avec une augmentation de $|Y^s|$. De même pour $p = 3$, mais seulement jusqu'à 49.33%. Pour $p = 4, 5$, UPool n'offre pas de meilleures performances que IF.

C.2.2 Approches par boîtes

Une boîte B est définie par un sous ensemble de points, une borne supérieure u dominant tous les points de la boite, et une borne inférieure l dominée par tous les points de la boîte. Soit $\mathcal{B} = \{B_1, ..., B_m\}$ une famille de boîte associée à $Y \subseteq \mathbb{R}^p$.

Definition C.8. *Soient $B_Y^i, B_Y^j \subseteq Y$ et $B_Z^k, B_Z^l \subseteq Z$, et leurs bornes inférieures et supérieures associées.* (B_Y^i, B_Z^k) box-to-box *domine* (B_Y^j, B_Z^l) *si et seulement si* $l_Y^i + l_Z^k \geqq u_Y^j + u_Z^l$

Observation C.2. *Si* (B_Y^i, B_Z^k) box-to-box *domine* (B_Y^j, B_Z^l)*, alors pour tout* $y \in B_Y^i \oplus B_Z^k$ *et tout* $y' \in B_Y^j \oplus B_Z^l$, $y \geqq y'$ *(see Figure C.4)*

C.2.2.1 Algorithme pour le filtrage par dominance par boîte

A chaque étape de filtrage intermédiaire, $\overrightarrow{Y^{s-1}}$, le résultat des étapes précédentes, et Y^s sont mis en boîte. On applique d'abord la dominance *box-to-box*, puis la dominance classique sur les combinaisons de points issues des boîtes restantes. Le filtrage *box-to-box* est réalisé de manière unidirectionnelle, sur la base du tri lexicographique décroissant des sommes de bornes supérieures des combinaisons de boîte, ordre qui préserve la dominance:

Observation C.3. *Soient $B_Y^1, B_Y^2 \subseteq Y$, et $B_Z^1, B_Z^2 \subseteq Z$. Si $u_Y^1 + u_Z^1 \succeq_{Lex} u_Y^2 + u_Z^2$, alors $l_Y^2 + l_Z^2 \not\geqq u_Y^1 + u_Z^1$*

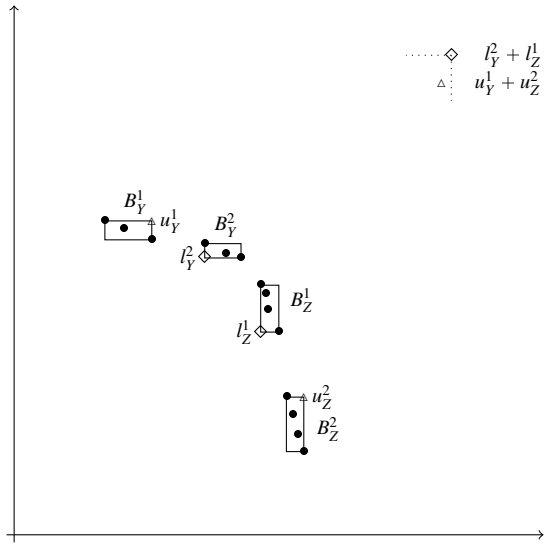


Figure C.4: Bi-objective illustration of *box-to-box* dominance.

C.2.2.2 Algorithme pour la création d'une famille de boîtes

Definition C.9. *Pour tous $y, y' \in Y$, y ε-domine y', noté $y \succeq_\varepsilon y'$ si et seulement si*

$$y_j(1+\varepsilon) \geqslant y'_j \quad \forall j \in \{1, ..., p\}$$

y est ε-indifferent à y', noté $y \sim_\varepsilon y'$ si et seulement si

$$y \succeq_\varepsilon y' \text{ and } y' \succeq_\varepsilon y$$

L'algorithme de mise en boûte considère un point $y \in Y$ et crée une boîte contenant y. Pour tout autre $y' \in Y$, on vérifie si $y \sim_\varepsilon y'$. Si oui, on ajoute y' à la boîte asocciée à y, et on supprime y' de Y. Sinon, on ne fait rien. Quand tout $Y \backslash \{y\}$ a été examiné, on supprime y de Y et on répète la procédure.

C.2.2.3 Résultats expérimentaux

Empiriquement, on constate que UPool n'est battu par la dominance *box-to-box* que pour $p = 4$, mais dans ce contexte il est aussi battu par IF. Donc les méthodes fondées sur les boîtes ne semblent pas viables.

C.3 Obtenir des ensembles bornant par décomposition

C.3.1 Ensembles bornants

Definition C.10. *Un point $y \in \mathbb{R}^p$ borne supérieurement $Y \subseteq \mathbb{R}^p$ si et seulement si*

$$\nexists y' \in Y, y' \succeq y$$

Un ensemble de points bornant supérieurement est appelé borne supérieure faible.

Definition C.11. *Un point $y \in \mathbb{R}^p$ borne inférieurement $Y \subseteq \mathbb{R}^p$ si et seulement si*

$$\nexists y' \in Y, y \succeq y'$$

Un ensemble de points bornant inférieurement est appelé borne inférieure faible.

Definition C.12. *(Ehrgott and Gandibleux (2001)) Pour $Y \subseteq \mathbb{R}^p$, UB est un ensemble bornant supérieurement fortement Y (pour la maximisation) si et seulement si*

$$\begin{cases} \forall y \in Y \; \exists y' \in UB, y' \succeq y \\ \forall y \in UB, y \text{ is an upper bound point to } Y \end{cases} \tag{C.2}$$

Definition C.13. *(Ehrgott and Gandibleux (2001)) Pour $Y \subseteq \mathbb{R}^p$, UB est un ensemble bornant inférieurement fortement Y (pour la maximisation) si et seulement si*

$$\begin{cases} \forall y \in Y \; \exists y' \in LB, y \succeq y' \\ \forall y \in LB, y \text{ is a lower bound point to } Y \end{cases} \tag{C.3}$$

Les ensembles bornant supérieurs sont les ensembles de points non-dominés de relaxations du problème original, et les ensembles bornant inférieurs, de restrictions du problème original.

C.3.2 Restrictions décomposables

Pour obtenir une restriction non-couplée du problème original, on fixe toutes, ou certaines variables couplées, à des valeurs admissibles ou neutres.

Definition C.14. *e_i est une valeur admissible pour x_i si il existe $x \in X$ tel que $x_i = e_i$, i.e. t.q. $(x_{-i}, e_i) \in X$.*

Definition C.15. *e_i est une valeur neutre pour x_i si pour tout $x \in X$, $(x_{-i}, e_i) \in X$.*

$x^s_K = (x^s_k \mid k \in K)$ dénote les variables du système s couplées par la contrainte couplante k. $x^s \sim (x^s_K, x^s_{\bar{K}})$, avec $x^s_{\bar{K}}$ le vecteur des variables non couplées de s. Soit $x_{\bar{K}}$ le vecteur des variables non couplées. On obtient la restriction *LocRes* et la borne inférieure associée en fixant toutes les

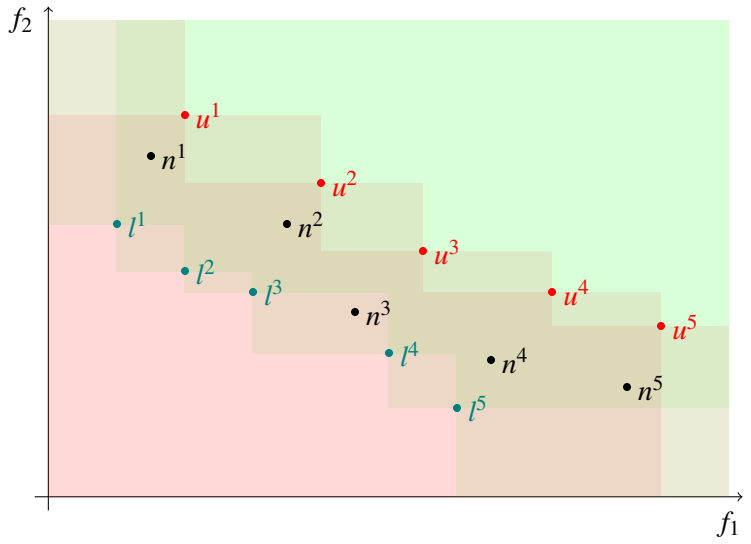


Figure C.5: $N = \{n^i \mid i = 1, ..., 5\}$, $L = \{l^i \mid i = 1, ..., 5\}$, $U = \{u^i \mid i = 1, ..., 5\}$. L borne inférieurement faiblement N, U borne supérieurement fortement N

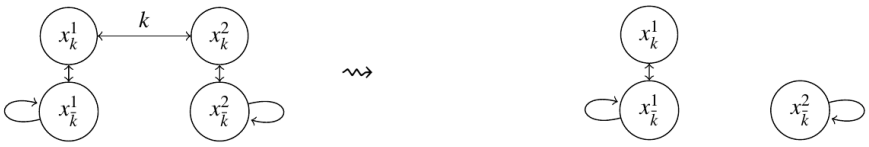


Figure C.6: Représentation graphique de la modification produisant la variante *restrict split*.

variables couplées. Dans la variante *restrict-split* on choisit pour chaque $k \in K$, un s couplé par k. On fixe les variables apparaissant dans la contrainte k, sauf celles de s.

Example C.2. *L'exemple suivant illustre la variante* restrict split *découplée du problème original, obtenue en fixant les variables des sous-systèmes autres que s_1 qui apparaissent aussi dans la contrainte couplate k.*

C.3.3 Relaxations décomposables

Une relaxation est obtenue en remplaçant les fonctions objectif f_j par des fonctions relâchées f'_j ou en considérant un sur-ensemble X' de X, obtenu en relâchant les contraintes de X. Le plus simple est d'ignorer ces contraintes. Pour obtenir la décomposabilité, l'on peut se limiter à ignorer les termes de fonctions de contraintes additives associées avec certains sous-systèmes. Ces contraintes sont "relâchabels" en ce sens. Par exemple les contraintes de budget linéaires sont relachable en ignorant un sous ensemble quelconque de variables.



Figure C.7: Représentation graphique de la modification produisant la variante *copy split*.

Example C.4.

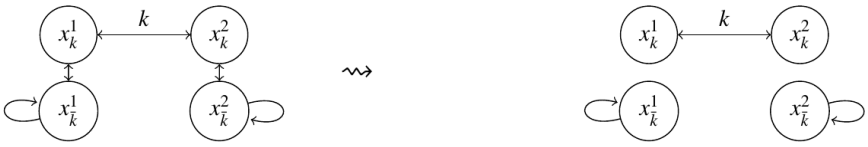


Figure C.8: Représentation gaphique de la modification produisant la variante *relaxation locale*.

C.3.3.1 Copy-splitting des contraintes couplantes

Si la contrainte couplante k est relâchable pour chaque $s \in \gamma(k)$, soit $g_k'^s(x_k^s) \leqslant b_k$ la relaxation de $g_k(x_k) \leqslant b_k$ associée à $s \in |\gamma(k)|$. Alors le problème suivant est une relaxation de (P), décomposable selon S.

$$\begin{array}{lll} \max & f(x) & \\ s.t. & g_k'^s(x_k^s) \leqslant b_k & \forall\, k \in K, s \in \gamma(k) \\ & x^s \in X^s & \forall\, s \in S \end{array}$$

Example C.3. *Le schéma suivant illustre le remplacement de la contrainte k par deux contraintes k' and k'', copies de k restreinte à chaque sous-système.*

C.3.3.2 Relaxations locales des variables couplées

On suppose que chaque $x^s \in X^s$ est du type $g^s(x^s) \leqslant b^s$. Pour $\bar{K}$ l'ensemble de variables non couplées $g^s(x^s) \leqslant b^s$ est relâchée en $g'^s(x^s_{\bar{K}}) \leqslant b^s$. On obtient:

$$\begin{array}{lll} \max & f(x) & \\ s.t. & x^k \in X^k & \forall\, k \in K \\ & g'^s(x^s_{\bar{K}}) \leqslant b^s & \forall\, s \in S \end{array}$$

Le problème est alors décomposé d'une part en la partie ne comprenant que les variables couplées, et d'autre part en chaque sous-système du problème original.

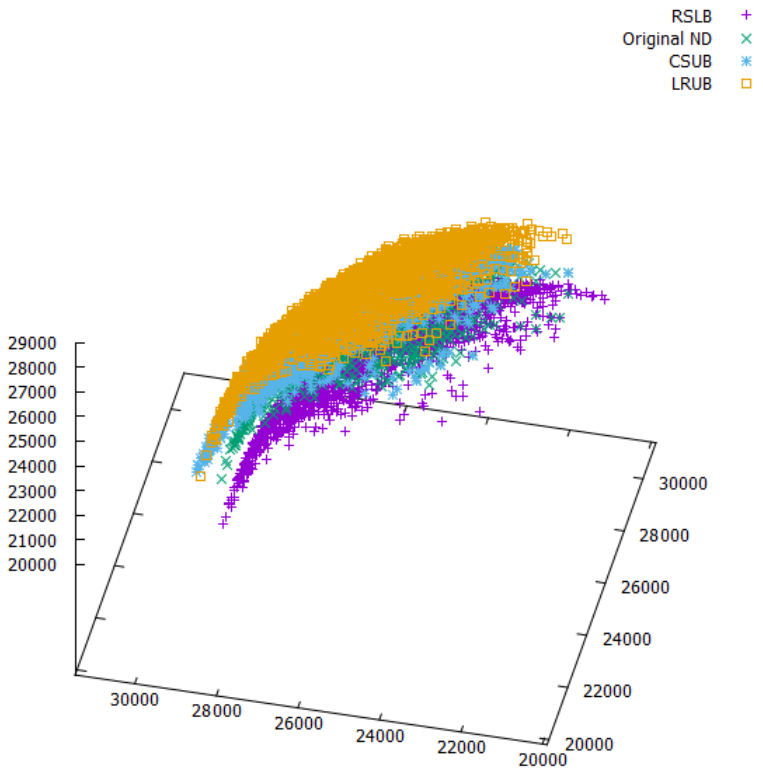


Figure C.9: Plot representation of the original non-dominated set and bound sets. Instance parameters are $p = 3$, $n = 75$, $|S| = 2$, $|K| = 1$, DV = 10

C.3.4 Résultats expérimentaux

On résout un problème générique couplé, puis on calcule à la fois des ensembles bornant supérieurs et inférieurs par décomposition. Le but est de comparer le temps de calcul nécessaire à l'obtention de ces bornes avec le temps de calcul nécessaire à la résolution du problème et d'estimer la qualité d'approximation de la solution du problème original permise par les ensemble bornants issues des relaxations et restrictions. La Figure C.9 montre l'encadrement de l'ensemble de solutions non-dominées original par des ensembles bornant inférieurs et supérieurs faibles.

Résoudre une variante restrict-split demande de produire une affectation des contraintes couplantes aux sous-systèmes. Ce choix est réalisé de manière heuristique en calculant:

$$s_k = \arg\max_{s \in \gamma(k)} \sum_{i \in k \cap s} \frac{\sum_{j=1}^{p} \pi_i^j}{c_i}$$

Concernant les contraintes relâchées de la variante copy-split, pour tout $k \in K$, si $\sum_{s \in S} \sum_{i \in s \cap k} a_i x_i \leqslant b_k$, alors, comme les a_i sont positifs, on a pour tout $s \in S$, $\sum_{i \in s \cap k} a_i x_i \leqslant b_k$. D'où l'expression:

$$
\begin{array}{lll}
\max & f(x) & \\
s.t. & \sum_{i \in s \cap k} a_i x_i \leqslant b_k & \forall\, s \in S, \forall k \in K \\
& \sum_{i \in s} c_i x_i \leqslant d_s & \forall\, s \in S \\
& x_i \in \{0,1\} & \forall\, i \in N
\end{array}
\tag{CS GCP}
$$

Les contraintes de la variante *locallement relachée* on considérant que du fait de la positivité des c_i on a $\sum_{i \in s \cap \bar{K}} c_i x_i \leqslant \sum_{i \in s} c_i x_i$, où $s \cap \bar{K}$ signifie $\setminus \bigcup_{k \in K} k$. Ainsi $\sum_{i \in s \cap \bar{K}} c_i x_i \leqslant d_s$ est une relaxation de $\sum_{i \in s} c_i x_i \leqslant d_s$, qu'on lui substitue dans la variante:

$$
\begin{array}{lll}
\max & f(x) & \\
s.t. & \sum_{s \in S} \sum_{i \in s \cap k} a_i x_i \leqslant b_k & \forall\, k \in K \\
& \sum_{i \in s \cap \bar{K}} c_i x_i \leqslant d_s & \forall\, s \in S \\
& x_i \in \{0,1\} & \forall\, i \in N
\end{array}
\tag{LR GCP}
$$

Pour mesurer la qualité d'approximation permise par un ensemble bornant, on utilise la notion de ε-dominance *a posteriori*:

Definition C.16. *A est une ε-approximation de B pour $\succeq_\varepsilon$ si et seulement si*

$$\forall y \in B\ \exists y' \in A, y' \succeq_\varepsilon y$$

Formellement, on calcule.

$$\varepsilon^*(A,B) := \min\{\varepsilon \in \mathbb{R} \mid \forall y' \in A\ \exists y \in B, y' \succeq_\varepsilon y\}$$

$\varepsilon^*(ND, L)$ donne la mesure de l'approximation de ND par la borne inférieure L, et $\varepsilon^*(U, ND)$ de l'approximation de ND par la borne supérieure U. On calcule aussi la proportion de l'ensemble non dominé du problème original déjà contenu dans l'ensemble bornant, dénoté $\frac{|ND \cap LB|}{|ND|}$ pour une LB.

In the following tables, *RS* denotes the *restrict split* restriction (Section 3.2.2) of the original problem, which provides a lower bound set. *CS* denotes the *copy split* relaxation (Section C.3.3.1) of the problem, which provides an upper bound set. *LR* denotes the relaxation obtained by omitting coupled variables in local subproblem constraints (Section C.3.3.2), which also provides an upper bound set. *T.* denotes computing time, $|.|$ denotes the size of a set of points, and $ApQ.(\varepsilon)$ denotes the quality of approximation in terms of *a posteriori* ε dominance.

Dans le cas bi-objectif, la solution est calculée rapidement, donc le coût du calcul des ensem-

bles bornant peut être considéré comme trop lourd. Au delà cependant, il devient négligeable.

Empiriquement, on observe que si le nombre de sous-systèmes augmente à nombre total de variables constant, le temps de calcul de toute variante décomposable diminue, puisque ces variantes se divisent en plus petit problèmes. Ainsi la borne inférieure associée à restrict-split est obtenue en moins de 2% du temps nécessaire à la résolution du problèmes quand $|S| = 4$. Une augmentation du nombre de contraintes couplantes à le même effet puisque une proportion supérieure des variables se trouvent fixées dans restrict-split. Quand $|K| = 1$, entre 17 et 26% de l'ensemble de points non dominés est déjà contenu dans la borne inférieure restrict-split pour $p = 3$, et entre 27 et 51% pour $p = 4$.

La borne supérieure copy-split paraît plus coûteuse que la borne inférieure restrict-split mais le gain de temps relativement à la résolution du problème original bénéficie aussi d'une augmentation de n et $|S|$, ainsi elle est obtenue en 1.2% du temps pour $p = 3$ et $|S| = 4$. Cette borne supérieure fournit une approximation d'excellente qualit, avec en moyenne 34% et 47% de la solution déjà contenue dans la borne supérieure pour $p = 3$, et entre 41% et 54% pour $p = 4$.

La borne supérieure issue de la relaxation locale est obtenue beaucoup plus rapidement que l'ensemble de solutions non dominées: en moins de 1 % du temps dans toutes les expériences. Toutefois, la qualité d'approximation qu'elle permet est largement moins bonne que celle la borne supérieure copy-split.

C.4 Problème d'application

C.4.1 Presentation of the REF problem

C.4.1.1 Formulation

REF est présenté comme un problème de production multi-site. Soit T un ensemble de *tâches*, S un ensemble de *sites* et M un ensemble de *machines*. $S(t)$ est l'ensemble des sites sur lesquels la tâche t peut être réalisée. $T(s)$ l'ensemble des tâches qui peuvent être réalisées sur le site s. $M(s) \subset M$ est l'ensemble des machines disponibles sur le site s, et par conséquent, $M(S(t))$ l'ensemble des machines sur lesquelles la tâche t peut être réalisée. $N := \{(t,m) \in T \times M \mid m \in M(S(t))\}$ dénote l'ensemble des variables, avec $|N| =: n$. L'affectation de t à m génère un profit g^j_{tm}, for $j \in \{1,...,p\}$, et consomme une quantité w_{tm} de ressource disponibel à chaque $s \in S$ tel que $m \in M(s)$. L'exemple C.5 fournit une représentation graphique de l'affectation de tâches aux machines.

Example C.5. *Admettons* $T = \{t_1, t_2, t_3\}$, $M = \{m_1, m_2, m_3, m_4\}$ *et* $S = \{s_1, s_2\}$, *tels que* $S(t_1) = \{s_1, s_2\}$, $S(t_2) = \{s_1\}$, $S(t_3) = \{s_2\}$, $M(s_1) = \{m_1, m_2\}$, $M(s_2) = \{m_3, m_4\}$. *Supposons que le budget de ressource de chaque site est de* 1.

	w_{tm}	t_1	t_2	t_3
s_1	m_1	0.6	0.5	
	m_2	0.5	0.5	
s_2	m_3	0.6		0.5
	m_4	0.5		0.6

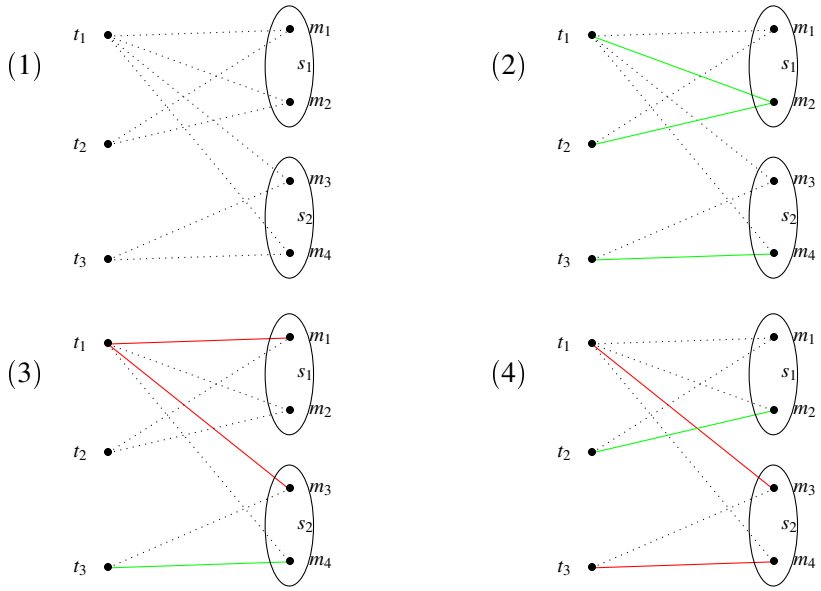


Figure C.10: Le graphe (1) représentes toutes les affecations possibles ainsi que les groupements de machines en sites. Le graphe (2) montre une solution réalisable, et les graphes (3) et (4) des solutions irréalisables

Soit $T_L = \{t \in T, |S(t)| = 1\}$ l'ensemble des tâches locales et $T_C = T \backslash T_L$ des tâches complexes. $T_L(s) = T_L \cap T(s)$ Le problème REF peut donc être écrit comme le programme mathématique suivant:

$$\max \sum_{s \in S} \sum_{t \in T(s)} \sum_{m \in M(S(t))} g_{tm}^{j} x_{tm} \qquad \forall\, j \in \{1, ..., p\}$$

$$s.t. \sum_{m \in M(S(t))} x_{tm} \leqslant 1 \qquad \forall\, t \in T_C \qquad (1)$$

$$\sum_{m \in M(S(t))} x_{tm} \leqslant 1 \qquad \forall\, s \in S, \forall\, t \in T_L(s) \qquad (2)$$

$$\sum_{t \in T(s)} \sum_{m \in M(S(t))} w_{tm} x_{tm} \leqslant b_s \qquad \forall\, s \in S \qquad (3)$$

$$x_{tm} \in \{0, 1\} \qquad \forall\, (t, m) \in N$$

C.4.1.2 Algorithme de programmation dynamique

Soit $\mathbf{0}_N$ la solution telle que pour tout $i \in N$, $\mathbf{0}_i = 0$, faisable par définition. $T = \{1, ..., \tilde{t}\}$, l'ensemble des tâches dans REF, correspond à l'nesemble des étapes de décision. Soit $\mathcal{Q}_0 := \{\mathbf{0}_N\}$ l'état initial du processus de décision. Les états des étapes suivantes associées à chaque $t \in T$ sont obtenus en construisant les solutions réalisables par affectation de chaque t aux machines $m \in M(S(t))$. $\tau(x, \delta)$ désigne la solution obtenue en prenant la séquence de décisions δ.

$$Comp(x) := \left\{\delta \in \Delta_t^{\tilde{t}} \mid \tau(x, \delta) \in X\right\}$$
$$Ext(x) := \{\tilde{x} \in X \mid \exists \delta \in Comp(x), \tilde{x} = \tau(x, \delta)\}$$

A l'étape finale de décision, l'ensemble des états est filtré par dominance selon la fonction $f = (f_1, ..., f_p)$, et aux étapes $t \in \{1, ..., \tilde{t} - 1\}$ selon $(f, \bar{w})$, où pour tout $x \in X$, $\bar{w}(x) = b - w(x) \in \mathbb{R}^{|S|}$, appelée capacité résiduelle, avec $b = (b_s | s \in S)$ la borne de budget de ressource pour s.

Observation C.4. *Pour tout $t \leqslant \tilde{t}$, $x, x' \in \mathcal{Q}_t$, tout $\delta \in Comp(x) \cap Comp(x')$,*

$$f_j(x) \geqslant f_j(x') \quad \Rightarrow \quad f_j(\tau(x, \delta)) \geqslant f_j(\tau(x', \delta)) \quad \forall j \in \{1, ..., p\}$$

and

$$\bar{w}_s(x) \geqslant \bar{w}_s(x') \quad \Rightarrow \quad \bar{w}_s(\tau(x, \delta)) \geqslant \bar{w}_s(\tau(x', \delta)) \quad \forall s \in S$$

La correction de l'algorithme de programmation dynamique est garantie par

Proposition C.3. *L'algorithme de programmation dynamique produit l'ensemble $\mathcal{Q}_{\tilde{t}}$ tel que*

$$\mathcal{Q}_{\tilde{t}} = \tilde{\mathcal{E}}(X, f)$$

Une instance d'un problème de programmatino dynamique est décrite par $(\mathcal{Q}_0, T_C, T_L, M, S)$, et on écrit $B \leftarrow DP(\mathcal{Q}_0, T_C, T_L, M, S)$ quand on obtient la préimage $\tilde{\mathcal{E}}(X, f)$ de $\mathcal{N}(f(X))$ par programmation dynamique.

C.4.1.3 Experimental results

On teste l'algorithme basique de programmation dynamique sur des instances où les taches complexes ont accès à tous les sites. Les budget de ressources des sites sont définis de manière à correspondre à des instances de sac à dos difficiles.

On trouve que l'algorithme de programmation dynamique est préférable à la méthode générique e-contrainte quand p augmente, mais pour $p = 2$, quand $|T_C|$ ou $|S|$ augmente, il devient beaucoup plus lent, du fait de la $p + |S|$ dimensionalité des états du processus de DP, qui induit de nombreuses incomparabilités et l'accumulation du nombre de solutions partielles.

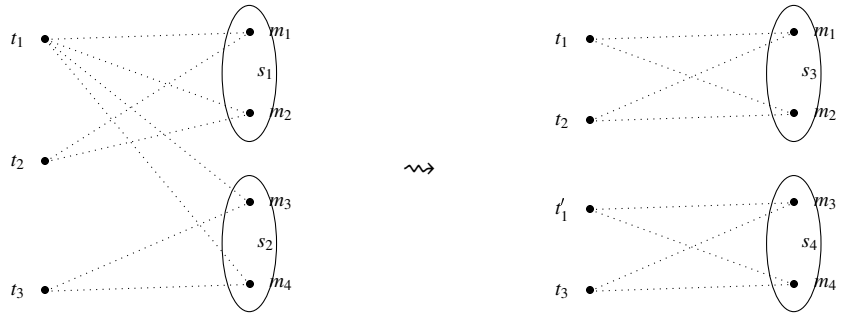


Figure C.11: Représentation graphique de copy split appliquée à REF

C.4.2 Ensembles bornants pour REF

C.4.2.1 Relaxation Copy-split

Dans le cas de REF, cette relaxation revient à remplacer chaque affectation de tâche complexe par $|S|$ copies de cette affectation, une chaque sous-système, comme illustré par la Figure C.11. On obtient alors un problèmes décomposable en $|S|$ problèmes de la forme suivante, pouvant être résolu par programmation dynamique.

$$
\begin{aligned}
\max \quad & \sum_{t \in T(s)} \sum_{m \in M(S(t))} g^j_{tm} x_{tm} && \forall\, j \in \{1, ..., p\} \\
s.t. \quad & \sum_{m \in M(s)} x_{tm} \leqslant 1 && \forall\, t \in T(s) \\
& \sum_{t \in T(s)} \sum_{m \in M(S(t))} w_{tm} x_{tm} \leqslant b_s \\
& x_{tm} \in \{0, 1\} && \forall\, t \in T(s),\ m \in M(S(t))
\end{aligned}
$$

C.4.2.2 Singleton borne supérieure forte.

On propose une heuristique gloutonne pour générer un point bornant fortement toutes les completions d'une solution partielle. Comme dans la relaxation copy-split, les tâches sont copiées, donnant un ensemble T' de tâches. Puis pour T' est remplacé par un ensemble de tâches "idéales" T^* tel que pour tous $t \in T'$, $m \in M(t')$,

$$
\begin{aligned}
g^{j*}_t &= \max_{m' \in M(t)} g^j_{tm'} \\
w^*_t &= \min_{m' \in M(t)} w_{tm'}
\end{aligned}
$$

Pour chaque $j \in \{1, ..., p\}$, on calcule la j-th composante du point bornant comme suit. Pour tous $s \in S$, on trie $T^*(s)$ en ordre décroissant de $\frac{g^{j*}_t}{w^*_t}$, et on réalise toutes les affectations possible

	1	2
g_t^{j*}/w_t^*	24 / 4	2/4

Figure C.12: Exemple de profits et poids associés à T^*

jusqu'à l'objet limite compris.

Proposition C.4. *$(u_1^*, ..., u_p^*)$ calculé de la manière précédemment décrite est une borne supérieure forte de l'ensemble des extensions d'une solution partielle.*

C.4.3 Approches "Bottom-up" fondées sur la programmation dynamique

C.4.3.1 Filtrage à l'aide de solutions précalculées

De manière similaire aux méthodes de Branch & Bound et à une approche proposée par Figueira et al. (2013), on compare un point bornant fortement supérieure les extensions d'une solution partielle à une solution réalisable précalculée. Si la solution précalculée domine cette borne, il est inutile de poursuivre les extensions de la soluiont partielle. Des expériences sont menées avec le solution de la variante restrict-split comme ensemble de solutions précalculées, et la borne supérieure forte singleton.

C.4.3.2 "Kickstarting"

Les sous-séquences d'étapes de décision associées à des tâches locales étant indépendantes les unes des autres, et l'ordre des étapes de décision étant indifférent, on peut séparer la sous séquence associée avec T_L en $|S|$ sous séquences associées avec $T_L(s)$ pour chaque $s \in S$. On résout $|S|$ problèmes de DP à $p+1$ critères, puis on combine et filtre les solutions avec $p+|S|$ critères, avant de finir la résolution du problème par DP.

C.4.3.3 Résultats expérimentaux

Empiriquement, l'algorithme de DP bénéfie le plus du filtrage par ensemble de solutions précalculées avec la variante restrict-split. Dans le cas $p = 2$, le résultat est ainsi obtenu jusqu'à 15.6 fois plus vite qu'avec l'algorithme basique, et jusqu'à 4 fois plus vite pour $p = 3$. Bien que l'application du Kickstarting permette une amélioration de la performance, elle est bien inférieure à celle obtenue avec le filtrage par ensemble précalculé.

De plus, la combinaison de cette dernière approche avec le Kickstarting amène à une diminution de la performance. En général, l'approche par programmation dynamique est très sensible à l'augmentation du nombre de critères, donc tout approche nécessitant de manipuler des états de taille $p + |S|$ à un moment donné du calcul est handicapée.

C.4.4 Approches "top-down" pour résoudre REF

C.4.4.1 Enumeration de toutes les manières de découpler REF

Chaque instance restrict-split de REF représente un manière de découpler le problème original en associant chaque tâche complexe à un sous-problème. Une telle association s'identifie à un élément de $\sigma \in S^{T_C}$. L'ensemble réalisable X^σ de toute variante restrict split est inclus dans X l'ensemble réalisable original.

Observation C.5. *Soit $(X^i \mid \in I)$ une famille d'ensembles avec $X^i \subseteq X$ pour tout $i \in I$, et $f : X \to \mathbb{R}^p$. Si $\bigcup_{i\in I} X^i = X$, alors*

$$\mathcal{E}(\bigcup_{i\in I} \mathcal{E}(X, f)) = \mathcal{E}(X, f)$$

Proposition C.5. *Pour une instance X et $\sigma \in S^{T_C}$ définissant une variante restrict REF with feasible set X^σ, $\bigcup_{\sigma\in S^K} X^\sigma = X$*

On résout donc REF par la simple procédure décrite par l'Algorithme 31:

Algorithm 31: Algorithme de résolution énumérant les découplages de REF

```
Out ← ∅
for σ ∈ S^{T_C} do
    Out ← Out ∪ Ẽ(X^σ, f)
return Ẽ(Out)
```

Empiriquement, on observe que même l'énumération de S^{T_C} peut être compétitive et plus rapide que l'exécution d'une méthode générique, quand le nombre de sous-système est relativement faible relativement à la taille des sous-systèmes.

C.4.4.2 Pre-computation of independent dynamic programming states

La variante restrict-split est résolue par décomposition, et chacun de ses sous-problèmes est résolu par programmation dynamique avec $p + 1$ critèrs seulement, et le filtrage des combinaison se limite à p critères. Ces variantes comprennent certaines tâches complexes devenues simples, mais les tâches déjà simple dans le problème original devraient être traitées de la même manière dans toutes les variantes. La sous-séquence de programmation dynamique associée avec ces tâches peut donc être précalculée et utilisée pour initialiser chaque variante découplée considérée. On considère l'ensemble réalisable X_L^s du sous-problème s réstreint à ses tâches locales, défini comme :

$$X_L^s := \{x \in \{0,1\}^{|T_L(s)|} \mid \sum_{t\in T_L(s)} \sum_{m\in M(s)} x_{tm} \leqslant 1 \quad \& \sum_{t\in T_L(s)} \sum_{m\in M(s)} w_{tm} x_{tm} \leqslant b_s\} \tag{C.4}$$

On calcule d'abord $Q_L^s = \tilde{\mathcal{E}}(X_L^s, (f^s, \bar{w}_s))$ pour tout $s \in S$. Puis tout tous $\sigma \in S^{T_C}$, on complète l'ensemble des $|S|$ sous-problèmes indépendants avec les tâches rendues simples pour chaque sous-problème par σ. On combine et filtre les solutions des sous-problèmes pour obtenir la solution de la variant restrict split. On calcule ensuite l'union des ensembles non-dominés pour tous les $\sigma \in S^{T_C}$, que l'on filtre.

Empiriquement, on observe que l'utilisatino de la programmation dynamique et du précalcul amène à une amélioration significative de la performance: pour des instances faiblement couplées, même dans le cas bi-objectif, le problème est résolu 4 à 8 fois plus vite qu'avec le l'algorithme générique. Pour $p = 3$, de 18 à 52 fois plus vite. Cette approche permet également une résolution beaucoup plus rapide qu'avec l'algorithme de programmation dynamique classique, même amélioré par le filtrage par une borne inférieure. Toutefois, si le degré de couplage de problème augmente (nombre de contraintes couplantes ou de sous-problèmes, donc valeur de S^{T_C}), la méthode générique redevient préférable dans le cas biobjectif.

C.4.4.3 Recherche arborescente et branching

Pour améliorer la méthode d'énumération des découplages, on la représente sous forme d'une arborescence, telle que chaque niveau correspond à une tâche complexe, et chaque décision de branchement à un sous-système auquel associer la tâche complexe. Les feuilles de cette arborescence correspondent à l'ensemble réalisable d'une variante restrict split du problème original, que l'on peut résoudre par décomposition et programmation dynamique.

Example C.6. *La Figure C.13 contraste branchement découplant et branchement binaire, classique en branch & bound.*

Un algorithme de branch & bound est correct si le schéma de branchement est exhaustif au sens suivant (ce qui est impliqué par la proposition C.5).

Definition C.17. *E is exhaustive if and only if for all $v \in V$*

$$\bigcup_{v' s.t. (v,v') \in E} X_{v'} = X_v \tag{C.5}$$

L'énumération est faite en profondeur d'abord. On calcul d'abord un ensemble de solutions candidates, qui correspondent à la solution de la variante restrict-split choisie par l'heuristique de la Section C.3.4, qui sera exclue de la recherche arborescente. On tire aussi partie du précalcul des sous-séquences de programmation dynamique décrit précédemment. La question est de savoir si des élimination de sous-arbre de type branch & bound peuvent être effectuées dans de telles arborescences.

Dans le branch & bound monocritère, il s'agit de montrer qu'une solution réalisable déjà connue est meilleure qu'une borne supérieure sur les solutions réalisables d'un sous arbre. La généralisation de cette notion au cas multiobjectif nécessite la définition de la "région de recherche", qui

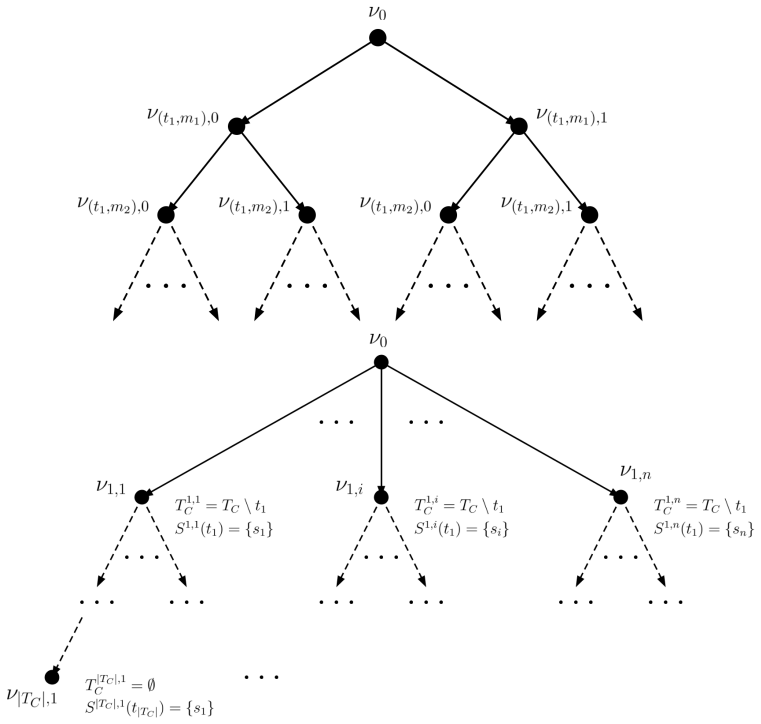


Figure C.13: Binary branching (top) versus decoupling branching (bottom)

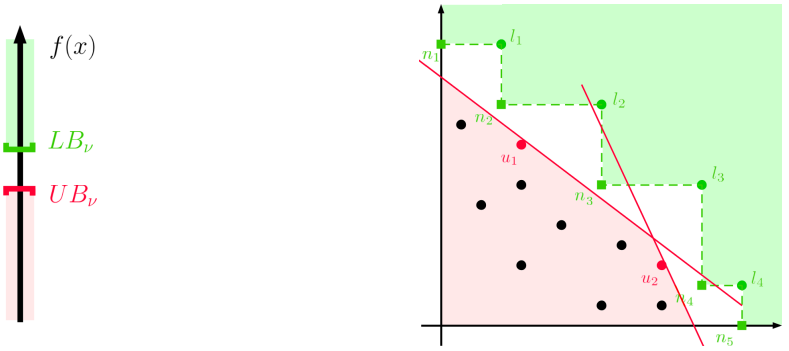
est l'ensemble des points non dominés par les solutions déjà connues, et est constituée de "zones de recherches" définies par des points charnières situés au minimum des paires de points réalisables déjà connus. En monocritère, cette région de recherche est simplement la demidroite réelle supérieure à la valeur de la meilleure solution connue.

Montrer que les solutions réalisables connues sont meilleures que les solutions d'un sous arbre revient à "séparer" la région de recherche de l'ensemble réalisable de ce sous arbre, ensemble réalisable borné par des ensembles bornant supérieurs, ou des hyperplans définis par des scalarisations, comme illustré dans la Figure C.14.

La séparation est réalisée en éliminant des zones de recherche à l'aide d'hyperplans définissant un demi-espace inaccessible. Si une zone de recherche est contenue dans ce demi espace, elle peut être supprimée du noeud actuel et de ses descendants. Ainsi la séparation n'a pas à être réalisée en un coup.

C.4.4.4 Application à la résolution de REF

On se limite dans cette expérience au cas biobjectif. L'arbre est exploré en profondeur jusqu'aux noeuds feuille, auxquels la variante restrict-split correspondante est résolue par programmation dynamique et décomposition. Aux noeuds intermédiaire, on génère une série d'hyerplans par la méthode dichotomique, pour essayer d'éliminer autant de zones de recherches actives que possible, en stoppant l'exploration à un certain seuil de précision défini par ε-dominance. La région de



(a) Single objective case

(b) Biobjective case

Figure C.14: Notion de région de recherche, de zone de recherche et de séparation illustrées dans le cas mono et multicritères. $\{n_1, ...n_4\}$ are the points defining the seach zones

recherche n'est pas mise à jour par la découverte de solutions réalisables aux problèmes feuilles, car cette opération est trop coûteuse et son implémentation doit être optimisée.

Pour toutes les instances testées à part une, l'approche branch & bound est légèrement plus rapide que l'approche énumérant les découplages avec décomposition, programmation dynamique et précalcul. Dans tous les cas, on voit qu'une proportion significative des noeuds feuilles de l'arborescence ont été évités. Des travaux supplémentaires sont nécessaires pour rendre cette exploration plus efficace et généraliser la séparation au delà de deux critères.

RÉSUMÉ

Un système complexe peut être vu comme une collection de sous-systèmes irréductiblement liés mais assez indépendants pour être distingués. Cette thèse considère l'optimisation de systèmes complexes dans le cadre multi-objectifs. L'interaction entre sous-systèmes d'un système complexe y prend la forme de contraintes couplantes, c'est-à-dire faisant intervenir des variables issues de différents sous-systèmes.
Après avoir rappelé les fondements de l'optimisation multi-objectifs et de l'algorithmique de filtrage par dominance, nous présentons la notion de système couplé, et définissons dans le cas multi-objectifs celle, centrale, de décomposition. Une implémentation simple de la décomposition suffit à améliorer le temps de résolution de problèmes non-couplés. Nous proposons de surcroit des méthodes algorithmiques avancées pour la combinaison de solutions de sous-problèmes et l'élimination des combinaison dominées, utilisant les notions de boîte bornante et d'algorithme unidirectionnel de filtrage par dominance.
Le défi principal de l'optimisation de systèmes complexes reste de prendre en compte les contraintes couplantes, tout en évitant de considérer l'entièreté du problème original en même temps. Nous proposons des restrictions et relaxations génériques des contraintes couplantes de problèmes couplés, permettant d'obtenir des ensembles bornant supérieurement et inférieurement l'ensemble de solutions non-dominées. Nous montrons que ceux-ci peuvent être calculés en tirant parti de la décomposition.
Enfin, nous présentons un problème d'application : une affectation multi-site multi-objectifs sous contraintes de ressources. Nous montrons que ce problème admet un algorithme de résolution par la programmation dynamique, et comment la décomposition peut être utilisée pour améliorer cette méthode initiale. D'une part, le processus séquentiel de décision peut lui-même être décomposé en sous-séquences indépendantes. D'autre part, des bornes ou des ensembles bornants obtenus par décomposition peuvent être utilisés pour accélérer le processus séquentiel de décision par l'élimination précoce de solutions partielles.

MOTS CLÉS

Optimisation Multiobjectifs, optimisation combinatoire, systèmes complexes, programmation dynamique, décomposition

ABSTRACT

A complex system is a collection of subsystems which are independent enough to be distinguished but linked together in significant ways. This thesis considers the optimization of complex systems within the framework of multiobjective optimization and focuses on a representation of complex systems as coupled systems, meaning that the interaction between subsystems is modeled by coupling constraints, i.e. constraints involving variables from different subsystems.
After having recalled the basic notions of multiobjective optimization and of dominance filtering algorithms, we introduce coupled problems, and define the key concept of decomposition in the multiobjective case. A simple implementation of decomposition already yields performance improvement in the resolution of uncoupled problems, but we provide further algorithmic improvements to the combination of solutions from subproblems and the elimination of dominated combinations, using notions of bounding boxes and of unidirectional dominance filtering algorithms.
Beyond the uncoupled case, the main challenge of complex systems optimization remains to take coupling constraints into account, while never having to consider whole complex system optimization problem at once. We propose generic restrictions and relaxations of coupling constraints, which yield upper and lower bounds set on the set non-dominated set of a coupled problem. We show that bound sets can be obtained using decomposition.
Finally, we present an application problem: a multiobjective multilocation assignment problem. We show that it admits a dynamic programming resolution method, and we show how decomposition can be used to improve on this initial resolution method. On the one hand, the sequential decision process can itself be broken down into independent subsequences. On the other hand, reasoning using bounds or bound sets obtained by decomposition can be used to speed up the sequential decision process by eliminating partial solutions early.

KEYWORDS

Multiobjecitve Opitmization, combinatorial optimization, complex systems, dynamic programming, decomposition