l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Audrey CERQUEUS

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 20 novembre 2015

Bi-objective branch-and-cut algorithms applied to the binary knapsack problem

surrogate bound sets, dynamic branching strategies, generation and exploitation of cover inequalities.

JURY

Président :	**M. Daniel VANDERPOOTEN**, Professeur, Université de Paris-Dauphine
Rapporteurs :	**M^{me} Kathrin KLAMROTH**, Professeur, Bergische Universität Wuppertal
	M. Thomas STUETZLE, Professeur, Université libre de Bruxelles
	M. Arnaud FRÉVILLE, Professeur, Université de Valenciennes et du Hainaut-Cambrésis
Directeurs de thèse :	**M. Xavier GANDIBLEUX**, Professeur, Université de Nantes
	M. Frédéric SAUBION, Professeur, Université d'Angers
Co-directeur de thèse :	**M. Anthony PRZYBYLSKI**, Maître de conférence, Université de Nantes

Remerciements

Je voudrais tout d'abord remercier les professeurs Kathrin KLAMROTH, Arnaud FRÉVILLE et Thomas STÜTILE, rapporteurs de ma thèse, pour leur temps et leurs retours sur mon travail. Je remercie également Professeur Daniel VANDERPOOTEN, non seulement pour avoir accepté d'être examinateur et président de ma soutenance de thèse, mais également pour avoir suivi mon travail régulièrement dans le cadre du comité de suivi de thèse. Je suis aussi reconnaissante du travail de suivi de thèse de Philippe DEPINCE.

Ma thèse n'aurait bien sûr pas été possible sans mes encadrants. Xavier GANDIBLEUX qui m'a fait confiance depuis le master et avec lequel j'ai eu des échanges scientifiques constructifs. Frédéric SAUBION, qui malgré le fait que nous ne soyons pas dans la même ville, a toujours été disponible pour discuter de nos travaux, avec l'optimisme qui le caractérise. Anthony PRZYBYLSKI pour son encadrement au quotidien et pour avoir eu l'occasion de tester ses talents culinaires.

Je tiens également à adresser mes remerciements à Stefan RUZIKA pour m'avoir accueilli dans son équipe de recherche pendant deux mois. En plus de son accueil plus que parfait, il m'a éclairé sur la partie "branch-and-cut" de cette thèse.

Je n'oublie pas Anne-Françoise, Elodie, Laurence, Christine, Annie, Jean-Paul, et bien d'autres, sans lesquels je me serais perdue dans les démarches administratives. Je les remercie également pour leur accueil et leur bonne humeur au quotidien.

Je souhaite aussi remercier les permanents de l'université de Nantes, Evgeny, Florian, Solène et tous les autres pour leurs échanges, aussi bien sur le plan scientifique, pédagogique et humain. Les doctorants ont également pris une place importante pendant ces trois années. Je remercie notamment ceux du bureau 218, pour leurs apports, forcés, à ma culture générale et leur soutien tout au long de ces années. Marie et sa volonté de toujours aider, qui mêlée à ses connaissances étendues en font presque une super-héroïne. Thomas parce qu'il est cool, même s'il distribue un certain nombre de "je n'aurai pas fait comme ça". Benjamin, grâce à qui mes talents de bricolage se sont révélés... inexistants et pour les délicieuses pauses thé. Je remercie Adrien, Ophélie et Liza pour le renfort chocolaté, ainsi que tous ceux que j'ai rencontrés tout au long de ma thèse : Nicolas, Soizic, David, Matthieu, Andreea, George, Marko, Nicolo et bien d'autres.

Enfin, mais pas des moindres je remercie ma famille et belle-famille pour leur soutien. Mes parents, mes frères et ma belle-sœur pour leurs conseils avisés et leur aide dans les moments difficiles, depuis toujours. Mes neveux et nièces pour leur tendresse. Alban pour avoir tout traversé avec moi, les moments difficiles et les joies, qui a toujours cru en moi et pour tout ce que je ne peux décrire.

P.S.: I would like to thank you, brave reader of this manuscript, please apologize my not so good english.

Contents

Résumé de la thèse

De nombreux problèmes réels se modélisent par un problème d'optimisation combinatoire. Par exemple, trouver le plus court chemin entre deux lieux peut être modélisé par un problème de cette classe. D'autre part, optimiser uniquement un objectif (par exemple la distance), ne suffit généralement pas à représenter un problème réel. Par exemple, une entreprise voulant maximiser son profit, peut aussi être intéressée par minimiser son impact environnemental. Plusieurs objectifs doivent alors être considérés. Si le décideur ne donne pas de préférence avant la résolution du problème, toutes les solutions présentant un compromis intéressant entre les différents objectifs doivent lui être retournées, afin qu'il établisse son choix. Dans ce contexte, nous considérons qu'une solution est un bon compromis entre les objectifs s'il n'est pas possible d'améliorer un objectif sans en dégrader un autre. Une telle solution est alors dite efficace.

Contexte : optimisation combinatoire multi-objectif

Les travaux de recherche effectués pendant cette thèse, s'attachent à la résolution de problèmes d'optimisation combinatoire multi-objectif (aussi appelé problèmes MOCO, de l'anglais *Multi-Objective Combinatorial Optimization*). Les problèmes d'optimisation combinatoire sont le sujet de recherche d'un grand nombre de travaux depuis plus d'un siècle, cependant l'intérêt accordé au cas multi-objectif est relativement récent (principalement depuis les années 90). Les problèmes d'optimisation combinatoire sont connus pour être particulièrement difficiles, en théorie comme en pratique. En effet, la plupart de ces problèmes sont $\mathcal{NP}$-difficiles, c'est-à-dire qu'il n'existe pas d'algorithme déterministe de complexité polynomiale pour leur résolution exacte (selon l'hypothèse que $\mathcal{P} \neq \mathcal{NP}$). Parmi les solutions efficaces des problèmes MOCO, nous pouvons distinguer les solutions supportées, qui sont solutions optimales pour une somme pondérée des fonctions objectif, et les solutions non-supportées. Ces dernières sont généralement plus difficiles à obtenir.

De nombreuses méthodes ont été développées pour résoudre, de manière exacte ou approchée, les problèmes MOCO. Les méthodes d'énumérations implicites des solutions, telles que les méthodes de *branch-and-bound* (méthodes de séparation et d'évaluation) et les méthodes de programmation dynamique, sont couramment utilisées pour la résolution exacte de problèmes MOCO. Dans ce manuscrit, nous nous intéressons plus particulièrement aux méthodes de *branch-and-bound* et, sans perte de généralité, nous considérons exclusivement des problèmes de maximisation.

Les méthodes de *branch-and-bound* partitionnent le problème en sous-problèmes, généralement en fixant une ou plusieurs variables. L'étape consistant à fixer une ou plusieurs variables est appelée la procédure de séparation, et le choix des variables à fixer est la stratégie de branchement. Ensuite, chacun des sous-problèmes créés est évalué. Une borne supérieure (un point dans l'espace des objectifs) ou un ensemble bornant supérieur (un ensemble de points dans l'espace des objectifs) est calculé, tel que les valeurs objectif des solutions admissibles du problème soient

inférieures ou égales à au moins un des points de cet ensemble. S'il est possible de prouver qu'aucune solution admissible pour le sous-problème ne peut être efficace, alors le sous-problème est sondé et un autre sous-problème ouvert (non-sondé) est choisi pour continuer la recherche. Dans le cas contraire, le sous-problème est de nouveau partitionné. Généralement, le calcul de l'ensemble bornant supérieur permet de générer de nouvelles solutions admissibles, soit directement, soit par réparation des solutions. La méthode s'arrête lorsque tous les sous-problèmes sont sondés.

Motivations

Le *branch-and-bound* est une méthodologie générique. Cependant, afin d'obtenir une méthode efficace en pratique, ses différents composants sont généralement instanciés spécifiquement pour un problème. Le calcul des bornes ou ensembles bornants supérieurs sont souvent basés sur une relaxation du problème et la relaxation utilisée dépend du problème considéré. La relaxation convexe, calculant seulement des solutions efficaces supportées, est généralement utilisée et offre de bonnes performances, lorsque la version mono-objectif du problème peut être résolue en complexité temporelle polynomiale ou pseudo-polynomiale en la taille de l'instance. La plupart des problèmes MOCO pour lesquels il existe un algorithme efficace en pratique appartiennent à cette catégorie de problèmes. La relaxation convexe est utilisée, par exemple, dans (Jorge, 2010). L'ensemble bornant obtenu en utilisant cette relaxation est l'ensemble bornant convexe le plus serré possible.

Cependant, lorsque la version mono-objective du problème ne peut être résolue en un temps raisonnable, la relaxation convexe devient coûteuse à calculer. Nous pouvons alors nous tourner vers des relaxations spécifiques au problème (comme la relaxation surrogate pour les problèmes de sac-à-dos multi-dimensionnel (Gandibleux and Perederieieva, 2011)) ou vers des relaxations génériques, telles que la relaxation continue par exemple. Cette dernière consiste à relâcher la contrainte d'intégrité des variables. Elle aboutit à un ensemble bornant moins serré que la relaxation convexe, mais peut être calculée facilement, par exemple par l'algorithme du simplexe multi-objectif. Afin de pallier la différence de qualité de l'ensemble bornant, nous pouvons introduire des inégalités valides (contraintes n'affectant pas l'ensemble des solutions admissibles, mais pouvant permettre de couper les solutions efficaces des relaxations), afin de resserrer l'ensemble bornant supérieur et donc de sonder plus tôt les sous-problèmes. Les algorithmes de *branch-and-cut* fonctionnent selon ce principe. S'ils ont beaucoup été étudiés pour des problèmes mono-objectifs, leur généralisation au contexte multi-objectif est très récente. (Jozefowiez et al., 2012) présente l'une des premières généralisations. Cependant la méthode présentée est une méthode ε-contrainte avec pas adaptatif, dont les problèmes mono-objectif sont résolus par un algorithme de *branch-and-cut* mono-objectif dédié.

Les composants des méthodes de *branch-and-bound*, autres que le calcul de l'ensemble bornant supérieur, ont été moins étudiés. Par exemple, l'ordre dans lequel les variables sont considérées pour le partitionnement est souvent statique. Si plusieurs stratégies ont été élaborées (Florios et al., 2010, Jorge, 2010), les comparaisons sont généralement effectuées sur une partie des stratégies et aucune d'entre elle ne semble obtenir de meilleurs résultats que les autres, sur l'ensemble des instances. Une stratégie, s'adaptant dynamiquement à l'instance, pourrait permettre d'améliorer les performances de la méthode.

Contributions

Les contributions de cette thèse s'appuient sur les observations présentées dans le paragraphe précédent. Le problème du sac-à-dos en variables binaires est le problème support de ces travaux. Le manuscrit s'articule de la manière suivante:

- Le Chapitre 1 présente l'état de l'art sur les problèmes MOCO. Les applications et les méthodes de résolution dédiées sont décrites.
- Le Chapitre 2 présente le problème du sac-à-dos. Plusieurs versions sont distinguées, en fonction du nombre de fonctions objectif et de contraintes prises en considération. Les difficultés propres à chacune de ces versions sont mises en exergue et les méthodes de résolutions dédiées sont développées.
- Le Chapitre 3 traite des stratégies de séparation pour les algorithmes de *branch-and-bound* appliqués au problème du sac-à-dos bi-objectif uni-dimensionnel.
- Le Chapitre 4 définit deux ensembles bornants basés sur la relaxation surrogate, pour le problème du sac-à-dos bi-objectif bi-dimensionnel. Des algorithmes permettant de calculer ces ensembles bornants sont également proposés.
- Le Chapitre 5 a pour but l'élaboration d'un algorithme de *branch-and-cut* pour le problème du sac-à-dos bi-objectif bi-dimensionnel.
- Enfin le dernier chapitre présente un résumé des contributions et présente des perspectives de travail.

Les Chapitres 3, 4 et 5, qui présentent les contributions de cette thèse, sont détaillés dans les prochaines sections.

Stratégies de branchement pour le sac-à-dos bi-objectif uni-dimensionnel

L'objectif du Chapitre 3 est d'analyser l'impact de l'ordre dans lequel les variables sont fixées dans une méthode de *branch-and-bound* et d'élaborer une stratégie efficace en pratique, s'adaptant à l'instance.

Dans un premier temps, nous présentons les stratégies de branchement que nous considérons dans ce travail. Certaines de ces stratégies sont issues de la littérature. Nous comparons l'efficacité de ces stratégies dans une méthode en deux phases pour laquelle la deuxième phase est une méthode de *branch-and-bound* (détaillée à la Section 3.2). Nous analysons aussi l'impact des pré-traitements adaptés de (Jorge, 2010) et (Delort, 2011) sur l'efficacité des stratégies. La Section 3.4 montre que la différence de performance entre les stratégies est davantage marquée lorsqu'aucun pré-traitement n'est appliqué. De plus, aucune stratégie n'aboutit aux meilleures performances, que les pré-traitements soient appliqués ou non.

Dans un second temps, nous nous attachons à réduire la taille des arbres de recherche en alternant les stratégies lors d'une même résolution. D'abord nous avons élaboré une méthode *oracle* évaluant chacune des stratégies, sur un pas de l'algorithme de *branch-and-bound*, et n'appliquant que la meilleure. Même si la méthode oracle permet de réduire les arbres de recherche obtenus de plus de 40%, le temps d'exécution de celle-ci est largement plus élevé que lorsqu'une unique stratégie statique est appliquée. L'analyse des exécutions de cette méthode oracle permet de réduire le nombre des stratégies statiques à considérer à 5 au lieu de 22. Malgré cette réduction, la méthode oracle reste, par nature, plus coûteuse qu'une stratégie de branchement statique.

Enfin, nous élaborons une stratégie de branchement dynamique, permettant de réduire la taille des arbres de recherche obtenus, tout en réduisant le temps de résolution du problème. À chaque séparation un sous-ensemble de 5 stratégies de branchement est considéré et la stratégie à appliquer est choisie à l'aide d'une méthode d'apprentissage par renforcement. La Section 3.6 présente et teste plusieurs méthodes d'apprentissage par renforcement. Celle donnant de meilleurs résultats

(Upper Confidence bound) permet de réduire le temps de résolution pour plus de la moitié des instances, par rapport à une stratégie statique.

Ensemble bornant supérieur basé sur la relaxation surrogate pour le problème du sac-à-dos bi-objectif bi-dimensionnel

Dans le Chapitre 4, nous définissons deux ensembles bornants supérieurs, basés sur la relaxation surrogate, pour le problème du sac-à-dos bi-objectif bi-dimensionnel. L'OSUB (pour Optimal Surrogate Upper Bound set) est l'ensemble bornant supérieur le plus serré qu'il est possible d'obtenir en utilisant la relaxation surrogate. L'OCSUB (pour Optimal Convex Upper Bound set) est l'ensemble bornant le plus serré possible basé sur la relaxation convexe de la relaxation surrogate (relaxation convexe surrogate) du problème. Pour obtenir ces ensembles bornants supérieurs, il est nécessaire d'utiliser plusieurs multiplicateurs pour la relaxation surrogate.

Dans un premier temps, nous tentons d'adapter la méthode dichotomique classique pour calculer l'OCSUB (Section 4.2), en calculant pour chaque itération le dual surrogate par l'algorithme présenté dans (Fréville and Plateau, 1993). Cependant un contre-exemple met rapidement en lumière que cette méthode ne permet pas d'obtenir l'OCSUB. Ensuite, un algorithme énumératif est développé (Section 4.4). Les multiplicateurs permettant d'obtenir tous les ensembles bornants issus de relaxation convexe surrogate sont énumérés. L'étude des relations de dominance entre les relaxations convexes surrogates (Section 4.5) permet de caractériser les multiplicateurs surrogates et d'élaborer un nouvel algorithme (*0M-M1 interval algorithm*), réduisant le nombre de multiplicateurs à considérer.

La Section 4.6 présente un algorithme d'approximation de l'OCSUB, basé sur les mêmes relations de dominance. Dans la Section 4.7, nous montrons que les algorithmes élaborés précédemment pour l'OCSUB peuvent être facilement adaptés à l'OSUB.

Les deux ensembles bornants et leurs calculs sont comparés expérimentalement sur un jeux d'instances dans la Section 4.8. Ils montrent que même si l'OSUB est considérablement plus serré que l'OCSUB, son calcul est également beaucoup plus coûteux. L'algorithme d'approximation de l'OCSUB et OSUB semble aboutir à un bon compromis entre le temps de calcul et la qualité de l'ensemble bornant obtenu.

Élaboration d'un algorithme de *branch-and-cut* pour la résolution du sac-à-dos bi-objectif bi-dimensionnel

Dans le Chapitre 5, nous nous attachons à élaborer un algorithme exact pour la résolution du sac-à-dos bi-objectif bi-dimensionnel.

Dans un premier temps, nous considérons un algorithme de *branch-and-bound* (Section 5.2). Nous comparons trois relaxations utilisées pour le calcul de l'ensemble bornant supérieur du sous-problème : la relaxation convexe, la relaxation surrogate et la relaxation continue (Section 5.2.2). La relaxation convexe donne l'ensemble bornant convexe le plus serré possible. Par contre, puisqu'il n'existe pas d'algorithme résolvant la version mono-objectif du problème en un temps raisonnable, la résolution de la relaxation convexe est coûteuse. Deux ensembles bornants basés sur la relaxation surrogate sont considérés, tous deux semblent trop coûteux pour être utilisés dans l'algorithme. La relaxation continue aboutit à un ensemble bornant largement moins serré que les deux autres relaxations et donc à des arbres de recherche considérablement plus larges. Cependant sa résolution est peu coûteuse et le temps de résolution du problème, en utilisant cette relaxation, est plus faible que pour les deux autres relaxations. Cette relaxation est donc utilisée. Afin de réduire la taille des arbres de recherche, et ainsi le temps de résolution, trois mécanismes sont mis en place : une stratégie de branchement basée sur des solutions efficaces extrêmes de la

relaxation (Section 5.2.3), une initialisation de l'ensemble bornant inférieur utilisant un opérateur de *path relinking* (Section 5.2.4) et l'introduction d'inégalités valides tout au long du processus de résolution. Nous nous attardons plus particulièrement sur ce dernier point.

Les inégalités valides que nous utilisons sont les inégalités de couverture (Crowder et al., 1983). Elles consistent à identifier un sous-ensemble de variables ne pouvant être sélectionnées ensemble, sans excéder une des contraintes de capacité du sac-à-dos. Dans le contexte mono-objectif, (Crowder et al., 1983) définissent une méthode permettant de générer des inégalités de couverture, violant la solution optimale de la relaxation continue, en se basant sur cette même solution (présentée dans la Section 5.3). Nous généralisons cette méthode au contexte multi-objectif (Section 5.4.2). Des inégalités de couverture sont alors générées à chaque nœud de l'arbre de recherche. Puisque les sous-problèmes diffèrent peu d'un nœud parent à un nœud enfant, les inégalités de couverture générées à un nœud sont adaptées pour ses nœuds enfants. Les expérimentations montrent que lorsque le nombre d'inégalités de couverture augmente, la taille des arbres de recherche diminue. Cependant, ce procédé étant coûteux, la résolution du problème est par conséquent plus coûteuse. Plusieurs variantes de la méthode sont proposées (Section 5.4.3), réduisant le nombre d'inégalités de couverture utilisées. Pour ceci, plusieurs stratégies sont élaborées : restreindre les solutions analysées afin de générer des inégalités de couverture et restreindre le nombre d'inégalités adaptées pour les nœuds enfants. Nous analysons également l'intérêt d'utiliser des inégalités de couverture étendues, c'est-à-dire pour lesquels les objets ayant un poids plus important sont également ajoutés à l'inégalité.

L'algorithme de *branch-and-cut* obtenu présente des résultats satisfaisants. En effet, il permet de réduire le temps de résolution pour plus de la moitié des instances et pour 10 des 13 instances comportant plus de 100 variables. Cependant, la méthode n'est pas compétitive avec la méthode ε-contrainte. Ceci est très certainement lié à un choix d'implémentation pour la résolution de la relaxation continue. En effet, afin d'éviter les instabilités numériques, nous avons choisi d'utiliser la méthode dichotomique. Nous envisageons d'utiliser la méthode du simplexe paramétrique, afin d'améliorerer les temps de résolution. La méthode de *branch-and-cut* devrait ainsi être compétitive.

Perspectives

Tout au long de ce manuscrit, nous analysons différents composants des algorithmes de *branch-and-cut*. Nous travaillons exclusivement sur deux problèmes supports : le sac-à-dos bi-objectif uni-dimensionnel et le sac-à-dos bi-objectif bi-dimensionnel. Une perspective de ce travail est d'appliquer les méthodes proposées dans cette thèse à d'autres problèmes MOCO. Nous pourrions, par exemple, généraliser aux problèmes comportant plus de deux fonctions objectif. Il faudrait alors utiliser des méthodes appropriées pour le calcul des ensembles bornants supérieurs (tel que(Przybylski et al., 2010b) pour la résolution de la relaxation convexe et de la relaxation surrogate convexe, et l'algorithme du simplexe multi-objectif (Ehrgott, 2005) pour la résolution de la relaxation continue).

Si le passage à plus de deux contraintes pour l'algorithme de *branch-and-cut* devrait être relativement direct, il n'en est pas autant pour l'OCSUB et l'OSUB. En effet les relations de dominance et la caractérisation des multiplicateurs pour la relaxation surrogate doivent être redéfinies dans cette situation.

Introduction

Combinatorial optimization problems allow to model and solve a variety of real life situations. For example, finding a route minimizing the distance can be modeled by a problem of this class. Nevertheless, considering only one objective to optimize may not be sufficient to represent the complexity of real life situations. Indeed, if a company is interested in maximizing its profit, it may also be interested in minimizing its ecological impact. Then several objectives have to be considered. If no preference is given a priori, all solutions such that it is not possible to improve an objective without degrading another one should be returned to the decision maker. After the solving process, the decision maker chooses among the returned solutions.

Multi-objective combinatorial optimization

The work presented in this manuscript deals with multi-objective combinatorial optimization problems. While single-objective combinatorial optimization problems are studied since more than a century, multi-objective combinatorial problems have gained interest since the 1990ies. Multi-objective combinatorial optimization problems are generally $\mathcal{NP}$-hard, meaning that there does not exist deterministic polynomial algorithms to solve them (under the assumption that $\mathcal{P} \neq \mathcal{NP}$). Various solution methods have been developed, both in exact approaches (guaranteeing the optimality of the solutions) and approximate approaches. Many implicit enumeration methods to exactly solve multi-objective combinatorial problems have been designed, such as branch-and-bound and dynamic programming methods. Branch-and-bound methods proceed by partitioning the problem into subproblems. Generally, the partitioning consists in fixing one variable; this processes is called the separation procedure. The subproblems created are evaluated. A point or set of points, called upper bound set, is computed such that the value on the objective function of the solutions of the considered subproblem cannot be better than the values of these points. If it is possible to prove, based on the upper bound set, that there cannot exist any solution in the subproblem, whose values are better than the values of the known solutions, this subproblem is pruned. Otherwise, the partitioning continues. It is easier to prune subproblems when feasible solutions of good quality are known. The set of feasible solutions found during the algorithm is the lower bound set and it can be initialized by a heuristic or metaheuristic method.

Even if the branch-and-bound method is a generic method, some of its components are often specialized to the considered problem. Indeed, this specialization allows to design methods that are efficient in practice. However, adapting those mechanisms to a different problem can be challenging. In this thesis, we will present new propositions for implicit enumeration methods to solve multi-objective combinatorial optimization problems, using generic methodologies. We also propose new strategies specifically for the knapsack problems.

Thesis guide

Chapter 1 is an introduction to combinatorial optimization. First, we present single-objective optimization problems and their multi-objective generalization. Definitions and notations related to multi-objective optimization are also presented. Finally, solution methods dedicated to multi-objective combinatorial optimization problems are described.

The knapsack problem will be used as a support problem. Several versions of this problem exist, depending on the number of objectives and dimensions considered. While the simplest versions, with one constraint and/or one objective, have been widely studied, versions with simultaneously several objectives and several dimensions remain challenging for exact solution methods. Chapter 2 describes those variants and their dedicated solution methods.

The method proposed, in this thesis (Chapter 3, 4, 5), to solve multi-objective combinatorial problems is a branch-and-cut method. A branch-and-cut method originates from branch-and-bound methods. However, in a branch-and-cut method, constraints (called valid inequalities) are added along the execution, in order to prune subproblems in the solving process. Thus, those two methods share many components such as the separation strategy and the upper bound set. In this thesis, we focus on the following components of the branch-and-cut method:

Separation strategy The choice of the variable used to partition the problem impacts the practical efficiency of the algorithm. Generally, the variables are considered in a static order in solution methods. However, this approach is often problem-specific. Moreover, its efficiency depends on the characteristics of the instance. In this thesis, we aim to define a dynamic strategy, that may adapt itself to the characteristics of the instance. In Chapter 3, we present a detailed comparison of the static branching strategies for the bi-objective knapsack problem and we introduce a new dynamic branching strategy, mixing different static strategies. The strategy is validated experimentally.

Upper bound set The evaluation of the subproblems is a key component of the implicit enumeration method. Indeed, when the quality of the evaluation increases, the probability of pruning a subproblem increases, too. However, the computational time required to compute the upper bound sets directly impacts the computational cost of the solution method. Thus, a tradeoff between these two aspects has to be found. In this thesis, we define two upper bound sets based on the surrogate relaxation (Chapter 4). We also design exact and approximation algorithms computing these bound sets. We prove their correctness and we assess their performance experimentally. Chapter 5 compares the performance obtained by different upper bound sets in a branch-and-bound method.

Initialization of solution Initializing the set of feasible solutions makes it possible to prune subproblems earlier in the solving process. In Chapter 5, we elaborate an initialization method, based on path relinking between supported efficient solutions of the problem.

Generation of valid inequalities Generating valid inequalities along the solving process allows to tighten the evaluation of the subproblems and thus to prune subproblems earlier. In Chapter 5, several implementations for the generation and exploitation of valid inequalities are experimentally compared and the most efficient version of the branch-and-cut method is compared to the branch-and-bound method and to the ε-constraint method.

A summary of these contributions is discussed in the last chapter, along with some perspectives for future research.

Publications

The contributions proposed in this thesis have been presented in scientific communications and articles.

The contribution on the dynamic branching strategy presented in Chapter 3 has been presented in a french conference.

– A. Cerqueus, X. Gandibleux, A. Przybylski and F. Saubion, "Efficacité des heuristiques de branchement pour le branch-and-bound multi-objectif : vers une gestion plus dynamique" in *15e Conférence ROADEF de la société Française de Recherche Opérationnelle et Aide à la Décision*, 2014, Bordeaux, France

The elaboration of upper bound sets based on the surrogate relaxation has been the subject of two scientific presentations and one article in an international journal of rank A.

– A. Cerqueus, A. Przybylski and X. Gandibleux, "Surrogate upper bound sets for bi-objective bi-dimensional binary knapsack problems" in *European Journal of Operational Research*, volume 244(2), pages 417-433, July 2015
– A. Cerqueus, X. Gandibleux and A. Przybylski, "Surrogate-based algorithm for computing an upper bound set for the 0/1 bi-objective bi- dimensional knapsack problem" in *22nd International Conference on Multiple Criteria Decision Making*, 2013, Málaga, Spain
– A. Cerqueus, X. Gandibleux and A. Przybylski, "Ensemble bornant supérieur basé sur la relaxation surrogate pour un sac-à-dos bi-objectif bi-dimensionnel" in *14e Conférence ROADEF de la société Française de Recherche Opérationnelle et Aide à la Décision*, 2013, Troyes, France

The work presented in Chapter 5, dealing with the elaboration of a bi-objective branch-and-cut method, has been presented at two international events. We can note that this work has been initiated during a two months visit at the university of Koblenz-Landau in Germany.

– A. Cerqueus, X. Gandibleux, A. Przybylski, S. Ruzika, and F. Saubion, "A branch-and-cut for the bi-objective bi-dimensional knapsack problem" in *23rd International Conference on Multiple Criteria Decision Making*, 2015, Hamburg, Germany
– A. Cerqueus, X. Gandibleux, A. Przybylski, S. Ruzika, and F. Saubion, "A branch-and-cut for the bi-objective bi-dimensional knapsack problem" in *Workshop on Recent Advances in Multi-Objective Optimization*, 2015, Nantes, France

Notations

The two following tables present respectively the notations and abbreviations used in this manuscript.

Notation	Signification
$\mathbb{R}, \mathbb{Z}, \mathbb{N}$	Set of real numbers, integers or positive integers
$\mathbb{R}^n, \mathbb{Z}^n, \mathbb{N}^n$	Set of vectors of n real numbers, integers or positive integers
$\mathbb{R}^n_+$	Set of vectors of n positive real numbers
n, m, p	Number of variables, constraints and objectives
v^T	Transposed of the vector v
$x = (x^1, \ldots, x^n)^T$	Solution of a problem
$y = (y^1, \ldots, y^p)^T$	Vector of objective values
$z(x) = (z_1(x), \ldots, z_p(x))^T$	Vector of objective functions
$y^1 \geqq y^2$	$y^1_k \geq y^2_k$ for $k = 1, \ldots, p$, y^1 weakly dominates y^2
$y^1 \geq y^2$	$y^1 \geqq y^2$ and $y^1 \neq y^2$, y^1 dominates y^2
$y^1 > y^2$	$y^1_k > y^2_k$ for $k = 1, \ldots, p$, y^1 strongly dominates y^2
$\mathbb{R}^p_{\geqq}, \mathbb{R}^p_{\geq}, \mathbb{R}^p_{>}$	$\{y \in \mathbb{R}^p : y \geqq 0\}, \{y \in \mathbb{R}^p : y \geq 0\}, \{y \in \mathbb{R}^p : y > 0\}$
X	Set of feasible solutions
Y	Image of X in objective space
X_E, Y_N	Set of efficient solutions, set of nondominated points
X_{SE}, Y_{SN}	Set of supported efficient solutions, set of supported nondominated points
X_{SE1}, Y_{SN1}	Set of extreme supported efficient solutions, set of extreme supported nondominated points
X_{SE2}, Y_{SN2}	Set of non-extreme supported efficient solutions , set of non-extreme supported nondominated points
X_{NE}, Y_{NN}	Set of non-supported efficient solutions, set of non-supported nondominated points
$X_{E_m}, X_{SE_m}, X_{SE1_m}, X_{SE2_m}, X_{NE_m}$	Minimal complete set of respectively X_E, X_{SE}, X_{SE1}, X_{SE2}, X_{NE}
$X_{E_M}, X_{SE_M}, X_{SE1_M}, X_{SE2_M}, X_{NE_M}$	Maximal complete set of respectively X_E, X_{SE}, X_{SE1}, X_{SE2}, X_{NE}
$\bar{Y}_N$	Set of nondominated point of a subproblem
conv S	Convex hull of S
$\triangle(y^r, y^l)$	Triangle defined by the points y^r, y^l and (y^l_1, y^r_2)

Abbreviation	Signification
MP	Mathematical programming
LP	Linear programming
ILP	Integer linear programming
MILP	Mixed integer linear programming
CO	Combinatorial optimization
MOP	Multi-objective optimization problem
MOLP	Multi-objective linear programming
MOILP	Multi-objective integer linear programming
MOMILP	Multi-objective mixed integer linear programming
MOCO	Multi-objective combinatorial optimization
KP	Single-objective uni-dimensional knapsack problem
$mDKP$	Single-objective multi-dimensional knapsack problem, with m dimensions
$pOKP$	Multi-objective uni-dimensional knapsack problem, with p objectives
$pOmDKP$	Multi-objective multi-dimensional knapsack problem, with p objectives and m dimensions
SR	Surrogate relaxation
SD	Surrogate dual
OSUB	Optimal surrogate upper bound set
OCSUB	Optimal convex surrogate upper bound set

1

Multi-objective combinatorial optimization

This chapter deals with mathematical programming problems and in particular those whose objective functions and constraints are linear and whose variables can only take discrete value. The formalism of single objective problems is described. Definitions and notations are then presented for multi-objective problems. The notions of bound and relaxation, which constitute the core of this thesis, are presented in this multi-objective context. The different classes of solutions are also defined. Finally, the main solution methods for multi-objective combinatorial optimization problems are presented.

1.1 Single-objective optimization

1.1.1 Mathematical programming

Mathematical programming aims to formalize and solve optimization problems, often encountered in real life, such as scheduling of a production line, selection of investments, etc. The decisions are modeled by variables and the constraints by equalities or inequalities over these variables. The decision maker aims to find, if it exists, an optimal solution according to an objective function of the variables. A mathematical programming problem can be formulated as follows:

$$\begin{array}{lll} \textit{opt} & z(x) & \\ \text{s.t.} & a_i(x) \Delta b_i & i = 1, \dots, m \\ & \Delta_i \in \{\leq, =, \geq\} & i = 1, \dots, m \\ & x \in \mathcal{D} \subseteq \mathbb{R}^n & \end{array} \tag{MP}$$

In this manuscript, the i-th component of a vector v will be denoted v_i and the vectors will be numbered using superscripts.

n is the number of decision variables of the problem. $\mathcal{D}_j$ is the *domain* of the variable x_j, i.e. the set of possible values for x_j, $j \in \{1, \dots, n\}$. The problem is composed of m constraints; each constraint is defined by a function $a_i : \mathbb{R}^n \to \mathbb{R}$ on the variables, a constant $b_i \in \mathbb{R}$ and a binary operator $\Delta_i \in \{\leq, =, \geq\}$, $i = 1, \dots, m$.

An assignment of a value for every $x_j \in \mathbb{R}$, $j = 1, \ldots, n$, is called a *solution*. A solution x is *feasible* if $x_j \in \mathcal{D}_j$ for all $j \in \{1, \ldots, n\}$ and x satisfies the set of constraints. The set of the feasible solutions of a problem is the set X.

$z : \mathbb{R}^n \to \mathbb{R}$ is the objective function to optimize. The term *"opt"* stands for the maximization or minimization of the objective function. Since the maximization and the minimization cases are symmetrical, without lost of generality, we will only consider in this manuscript maximization problems.

Among the feasible solutions, we are particularly interested in those optimizing the objective function, i.e. solutions $x^* \in X$ such that for all $x' \in X$ we have $z(x^*) \geq z(x')$. Such a solution is called an *optimal* solution. Generally z is not injective, thus there might exist several optimal solutions for a same problem. For some problems, there does not exist any optimal solution, in particular if the problem is infeasible ($X = \emptyset$) or if z is not bounded on X.

1.1.2 Linear programming

Linear programming (LP) is a particular case of mathematical programming. In LP problems, the objective function and the constraints are linear. In its standard form, a linear programming problem considers only non-negative values for the variables. In this manuscript, all LP problems are stated according to the standard form.

The objective function can then be written $z(x) = \sum_{i=1}^{n} c_j \, x_j$ where c_j is the *cost* (or *profit*) associated to the variable x_j, $j \in \{1, \ldots, n\}$. Alternatively, the objective function can be defined as the product $c^T \, x$ (where c^T is the transpose of c and $c^T = (c_1, \ldots, c_n)$).

Thanks to the linearity of the constraints, the coefficients of the variables on the constraints can be represented by a matrix $A \in \mathbb{R}^{m \times n}$, called the *constraint matrix* and the right hand sides of the constraints by $b \in \mathbb{R}^m$. The variables can be bounded.

A linear programming problem is thus formulated, in its canonical form, as:

$$\begin{array}{ll} \max & c^T \, x \\ \text{s.t.} & A \, x \leq b \\ & x \in \mathcal{D} \subseteq \mathbb{R}^n_+ \end{array} \qquad \text{(LP)}$$

The feasible set X is either:
– a polytope,
– an unbounded polyhedron,
– the empty set.

In all cases, the feasible set X is convex. Since z is linear, this geometric characterization allows to locate an optimal solution on one of the vertices of X.

Chvátal (1983), Schrijver (1986) and Teghem (2003) present solution methods for LP, such as the simplex method and the interior point method. The simplex method is based on the characterization of the optimal solution.

1.1.3 Integer linear programming

Integer linear programming problems (ILP) are linear optimization problems for which the variables are restricted to integer values. For example, the variables can represent the number of

persons to affect to a job, the selection of a path in a graph, etc. An ILP problem can be defined as follows:

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & A\,x \leq b \\ & x \in \mathcal{D} \subseteq \mathbb{N}^n \end{array} \tag{ILP}$$

The coefficients of the variables in matrix A and in the vector c have generally integer values. In this manuscript, we consider integer coefficients for ILP problems.

The integrality of the variables implies that the feasible set X is no longer convex and there is no straightforward characterization of the optimal solutions. Thus the solving of an ILP is more difficult than for a LP. The solution methods used for ILP problems are different from LP problems. Implicit enumeration such as dynamic programming methods or branch-and-bound methods; and cutting plane algorithms are commonly used (Wolsey, 1998).

Constraint programming solvers and SAT-solver consider ILP problems. However, these solvers focus more on finding a solution satisfying the constraints than on the optimization part.

Note that a problem can also involve integer and continuous variables. It is then called a *mixed integer linear programming* (MILP) problem. The solution methods for this class of problems are similar as the one for ILP problems.

1.1.4 Combinatorial optimization

Combinatorial optimization problems (CO) constitute a subclass of ILP with the particularity that the constraint matrix has a combinatorial structure. Classically the variables are binary.

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & A\,x \leq b \\ & x \in \{0,1\}^n \end{array} \tag{CO}$$

where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$.

Knapsack problems, assignment problems, shortest path problems and vehicle routing problems are few of the reference CO problems.

Because of the combinatorial structure, CO problems are often $\mathcal{NP}$-hard (Garey and Johnson, 1979). Solution methods are generally dedicated to a particular CO problem and exploit the structure of the problem (Wolsey and Nemhauser, 1999).

1.2 Multi-objective optimization

When solving a single-objective optimization problem, the decision maker aims to optimize one unique objective (for example maximizing the incoming). However, the complexity of real life situations may not be represented only by one objective. For example, the decision maker could be interested in maximizing the profit of its production line while minimizing the environmental impact. Therefore, several objective functions have to be considered in order to formalize more accurately the expectation of the decision-maker; this is the scope of multi-objective optimization.

1.2.1 Definition

Multi-objective optimization problems (MOP) are defined to optimize p objective functions:

$$\begin{array}{rl} \text{“max”} & z(x) = (z_1(x), \ldots, z_p(x)) \\ \text{s.t.} & x \in X \end{array}$$

Even if the constraints and domains are not expressed explicitly in this formulation, they are implicitly given by $x \in X$. $z : \mathbb{R}^n \to \mathbb{R}^p$ is the function associating to a solution x the vector of objective values $(z_1(x), \ldots, z_p(x))$. $z(x)$ is called a point. $Y = z(X)$ is the image of the feasible set X by z.

The different classes of multi-objective problems correspond to the classes of single-objective problems: multi-objective linear programming (MOLP), multi-objective integer linear programming problem (MOILP), multi-objective mixed integer linear programming problem (MOMILP) and multi-objective combinatorial optimization (MOCO). Note that the characterizations of the feasible sets presented for single-objective problems still hold for their multi-objective version.

The objective functions can be considered sequentially or simultaneously. In the following we will consider them simultaneously. The objective functions considered in a multi-objective optimization problem are generally conflicting. Thus there generally does not exist a single feasible solution optimizing simultaneously all the objective functions. The notion of optimality, as previously defined for single-objective optimization problems, does not handle multi-objective optimization problems. Next section defines the meaning of the maximization operator for multi-objective problems.

1.2.2 Solutions of multi-objective optimization problems

From now on, we distinguish the decision space $\mathbb{R}^n$ (domain of the decision variables) and the objective space $\mathbb{R}^p$ (image of the solutions).

Dominance and efficiency

While solutions of a single-objective problem can easily be compared (the value according to the objective function is a scalar), this comparison has to be refined for two or more objective functions. Definition 1 presents the notion of dominance (also called Pareto dominance (Pareto, 1896)) , which is a componentwise order stipulating if a solution is “better” than another with respect to the $p \geq 2$ objective functions.

Definition 1 (Dominance)**.** *Let us consider two points* $y^1, y^2 \in \mathbb{R}^p$.

– y^1 weakly dominates y^2*, denoted by* $y^1 \geqq y^2$*, if*

$$y^1_k \geq y^2_k \text{ for } k = 1, \ldots, p.$$

– y^1 dominates y^2*, denoted by* $y^1 \geq y^2$*, if*

$$y^1 \geqq y^2 \text{ and } y^1 \neq y^2$$

– y^1 strictly dominates y^2*, denoted by* $y^1 > y^2$*, if*

$$y^1_k > y^2_k \text{ for } k = 1, \ldots, p.$$

Two points are *incomparable* according to this componentwise order if there does not exist any dominance relation between them (i.e. if none of them weakly dominates the other). Example 1 illustrates the Definition 1.

Example 1. *Let us consider the three points* $(3,7),(5,6)$ *and* $(6,7)$. $(6,7)$ *dominates, and thus also weakly dominates,* $(3,7)$. $(6,7)$ *strictly dominates* $(5,6)$ *(and also dominates and weakly dominates it).* $(3,7)$ *and* $(5,6)$ *are incomparable.*

Definition 1 allows us to define the dominance cones: $\mathbb{R}^p_{\leqq} = \{y \in \mathbb{R}^p : y \leqq 0\}$, $\mathbb{R}^p_{\leq}$ and $\mathbb{R}^p_{<}$ are defined analogically. $y + \mathbb{R}^p_{\leqq}$ stands for the vector addition of $y \in \mathbb{R}^p$ and any vector of $\mathbb{R}^p_{\leqq}$, it thus defines the area whose points weakly dominate y. This notation is also extended to $S + \mathbb{R}^p_{\leqq}$ with $S \subset \mathbb{R}^p$, the other dominance cones ($\mathbb{R}^p_{\leq}$ and $\mathbb{R}^p_{<}$) and to the vector subtraction.

When solving a multi-objective problem we aim to obtain solutions such that it is not possible to improve one of the objective functions without degrading another. Such a solution is called an efficient solution (see Definition 2).

Definition 2 (Efficient solution and nondominated points)**.** *Let us consider* $\hat{x} \in X$.

- $\hat{x}$ *is* efficient *if there does not exist* $x \in X$ *such that* $z(x) \geq z(\hat{x})$. $z(\hat{x})$ *is said to be* nondominated.
- $\hat{x}$ *is* weakly efficient *if there does not exist* $x \in X$ *such that* $z(x) > z(\hat{x})$. $z(\hat{x})$ *is said to be* weakly nondominated.

Definition 3 (Efficient set and nondominated set)**.** *The set of efficient solutions* $X_E \subset X$*, also called* efficient set*, is the set* $\{x \in X :$ *there does not exist* $x' \in X, z(x') \geq z(x)\}$*. Its image in objective space, i.e.* $z(X_E)$*, is called the* nondominated set Y_N.

An alternative definition of Y_N can be given using the dominance cones: $Y_N = \{y \in Y : ((y + \mathbb{R}^p_{\leqq}) \cap Y) = \{y\}\}$. We extend this notation to any $S \subset \mathbb{R}^p$, $S_N = \{s \in S : (s + \mathbb{R}^p_{\leqq}) \cap S = \{s\}\}$.

Definition 4 (Equivalent solutions and complete set (Hansen, 1980))**.** *Two feasible solutions* x^1 *and* x^2 *are* equivalent *if* $z(x^1) = z(x^2)$.

- *A* complete set *of efficient solutions is any subset* $X' \subseteq X$ *such that* $z(X') = Y_N$.
- *A* minimal complete set *is a complete set without any equivalent solutions. Any minimal complete set of efficient solutions is denoted by* X_{E_m}.
- *The* maximal complete set *is the complete set with all equivalent solutions, it is denoted by* X_{E_M}

Remark 1. *When considering two objective functions, the nondominated points can easily be ordered with decreasing values on the first objective function, while the values on the second objective function increase. This order is called the* natural order. *We will say that two points* y^1 *and* y^2 *are adjacent in the natural order of* Y_N *if there does not exist* $y \in Y_N$ *such that* $y^1_1 > y_1 > y^2_1$ *and* $y^1_2 < y_2 < y^2_2$.

Example 2 illustrates Definition 2 and Remark 1.

Example 2. *Figure 1.1 presents the dominated and nondominated points of a bi-objective combinatorial optimization problem. The points* y^l*, with* $l \in \{1,\dots,6\}$ *are ordered according to the natural order.* y^l *and* y^{l+1} *are adjacent in the natural order of* Y_N*, for* $l \in \{1,\dots,5\}$.

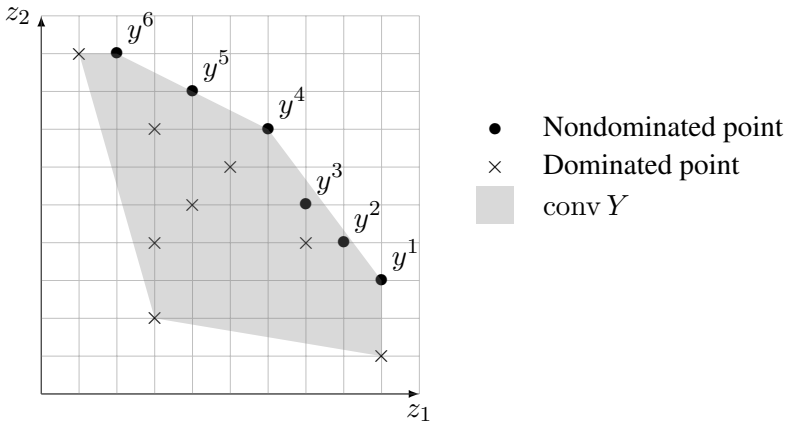


Figure 1.1: Illustration of the nondominated points in Y and the natural order in Y_N

Classification of efficient solutions

The classification of the efficient solutions is based on the weighted sum problem, defined by Geoffrion (1968). The weighted sum problem considers a scalarization of the objective function, aggregating them using a multiplier $\lambda \in \mathbb{R}^p_{\geq}$.

Theorem 1 ((Geoffrion, 1968)). *For a given multi-objective optimization problem P with p objective functions, P_λ is the weighted sum problem using weight vector λ:*

$$\max\{\lambda_1 z_1(x) + \cdots + \lambda_p z_p(x) : x \in X\} \qquad (P_\lambda)$$

Supposing x^* is the optimal solution of P_λ, then the following statements hold.
- *If $\lambda \in \mathbb{R}^p_{\geq}$ then x^* is weakly efficient.*
- *If $\lambda \in \mathbb{R}^p_{>}$ then x^* is efficient.*
- *If $\lambda \in \mathbb{R}^p_{\geq}$ and x^* is the unique optimal solution of P_λ then x^* is efficient.*

In this manuscript, the weight vector λ will be called a *direction* in the objective space.

An efficient solution x is called *supported* if there exists $\lambda \in \mathbb{R}^p_{>}$ such that x is optimal for P_λ. We denote X_{SE} the set of supported efficient solutions and $Y_{SN} = z(X_{SE})$ the set of supported nondominated points. The image of a supported efficient solution is located on the boundary of convex hull of Y. Let $S \subset \mathbb{R}^p$, conv S denotes the convex hull of S.

A non-supported efficient solution is an efficient solution x for which there does not exist $\lambda \in \mathbb{R}^p_{>}$ with x optimal for P_λ. Its image in objective space is called a non-supported nondominated point. X_{NE} and Y_{NN} are respectively the set of non-supported efficient solutions and non-supported nondominated points.

The set of supported efficient solutions can be split in two parts: the set of *extreme* supported solutions X_{SE1} and the set of *non-extreme* supported efficient solutions X_{SE2}. A supported efficient solution is extreme if its image in objective space corresponds to an extreme point of conv Y. The set of extreme supported nondominated points is denoted Y_{SN1}. It can be obtained by a dichotomic method which is presented in Section 1.4.1. Y_{SN2} is the set of non-extreme supported nondominated points.

Definition 4 can be extended to the sets X_{SE}, X_{SE1}, X_{SE2} and X_{NE} defining the sets X_{SE_m}, X_{SE_M}, X_{SE1_m}, X_{SE1_M}, X_{SE2_m}, X_{SE2_M}, X_{NE_m} and X_{NE_M}. The notion of adjacency defined for Y_N can be extended to Y_{SN}, Y_{SN1}, Y_{SN2} and Y_{NN}.

Figure 1.2 illustrates the definitions above.

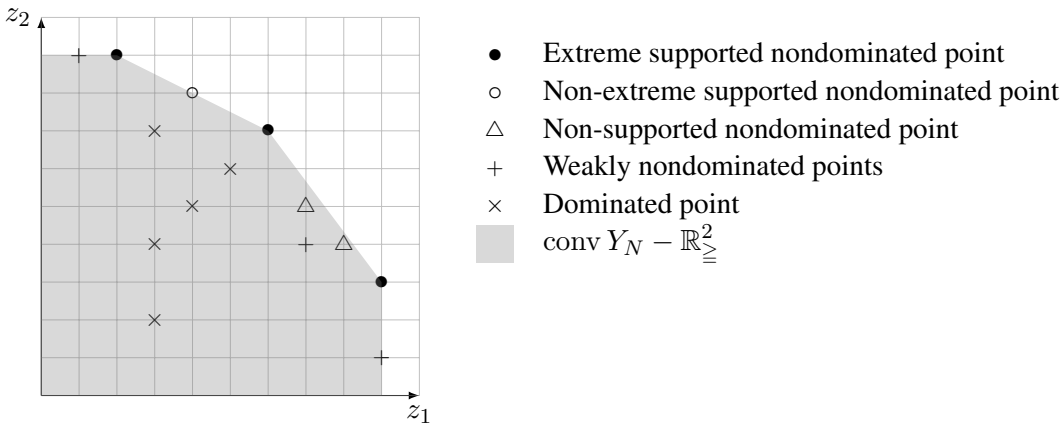


Figure 1.2: Illustration of the classification of points

Lexicographic order and lexicographic optimal

The dominance relation is a partial order on the points (and thus on the solutions), i.e. there exist incomparable points according to the dominance relation. On the contrary, the lexicographic order is a total order on the points.

Definition 5 (Lexicographic order)**.** *Let $y^1, y^2 \in \mathbb{R}^p$, then $y^1 >_{lex} y^2$ if there exists $l \in \{1, \dots, p\}$ such that for all $k < l$, $y^1_k = y^2_k$ and $y^1_l > y^2_l$.*

$y^1 >_{lex} y^2$ or $y^1 = y^2$ is denoted by $y^1 \geqq_{lex} y^2$.

Definition 6 (Lexicographic optimal)**.** *Let π be a permutation on the objective functions $\{1, \dots, p\}$. A feasible solution x is called* lexicographic optimal *according to π if for all $x' \in X$, $z_\pi(x) \geqq_{lex} z_\pi(x')$, where $z_\pi(x) = (z_{\pi_1}(x), \dots, z_{\pi_p}(x))$.*

A solution x is said to be lexicographic optimal *if there exists a permutation π such that x is lexicographic optimal according to π.*

1.2.3 Solving a multi-objective combinatorial problem

Let us consider two efficient solutions x^1 and x^2, by definition there does not exist any dominance relation between $z(x^1)$ and $z(x^2)$. But the decision maker could have a preference between x^1 and x^2. Indeed he might have preferences on the objectives or on the composition of the solutions. Based on this observation, there exist three contexts to solve a MOP (Evans, 1984):

- The *a priori* context, in which the decision maker specifies his preferences before solving.
- The *a posteriori* context, in which the decision maker chooses the preferred solution among the complete set of efficient solutions returned by the solution method.
- The *interactive* context, in which the solution method is adjusted according to the preferences given by the decision maker all along the solving process.

In this manuscript, we consider the a posteriori context.

We already mentioned in Section 1.1 that the difference in the nature of the variables changes the characterization of the feasible set and thus the characterization of the optimal solution. This observation is also valid in the case of multi-objective optimization.

The feasible set X of a MOLP problem is convex and continuous, by linearity of the objective functions so is Y. The vertices of Y correspond to vertices in X. Generally Y_N contains an infinite number of points, located on nondominated faces (of dimension 1 to $p-1$) defined by the extreme supported nondominated points (vertices of Y). The multi-criteria simplex algorithm (Ehrgott, 2005), the outcome-based exploration methods (Benson (1998), Ehrgott et al. (2012)) or dichotomic methods (Section 1.4.1) can be used to solve a MOLP.

The feasible set of a MOILP or a MOCO problem is a discrete and non convex set of points. Moreover, the set of nondominated points generally contains non-supported nondominated points, which are particularly difficult to obtain in a methodological point of view (since they are not optimal solutions of a weighted sum problem). Even if some CO problems are in $\mathcal{P}$, MOCO problems are generally $\mathcal{NP}$-hard, #P-complete and intractable (there might be an exponential number of efficient solutions) (Ehrgott, 2000). Methods to solve those problems are presented in Section 1.4.

1.2.4 Bounds and bound sets

Single-objective optimization

Solution methods for single-objective combinatorial optimization generally bracket the objective value of the optimal solutions using bounds (see Section 1.4).

Definition 7 (Upper and lower bound in single-objective optimization)**.** *Let x^* be an optimal solution of a problem P.*

- *An upper bound on $z(x^*)$ is a value $u \in \mathbb{R}$ such that $z(x^*) \leq u$.*
- *A lower bound on $z(x^*)$ is a value $l \in \mathbb{R}$ such that $l \leq z(x^*)$.*

The upper bound is often obtained by solving a relaxed version of the problem (see Section 1.3) and the lower bound is generally the objective value of the best known feasible solution.

We can note that the characterization of the upper and lower bounds are interchanged in the case of minimization problems.

Multi-objective optimization

In the multi-objective context, we aim to bound the set of nondominated points Y_N. Contrary to the single-objective case, there is no longer only one value to bound, but a set of points (i.e. vectors of p objective values). The direct transposition of the notion of bound in a multi-objective context consists in finding two points (a lower bound $l \in \mathbb{R}^p$ and an upper bound $u \in \mathbb{R}^p$), such that, for each objective function, the inequalities presented in Definition 7 are satisfied for each nondominated point of the problem.

Definition 8 (Upper and lower bound in multi-objective optimization)**.**

- *An upper bound on Y_N is a point $u = (u_1, \ldots, u_p) \in \mathbb{R}^p$ such that for all $y \in Y_N$, for all $k \in \{1, \ldots, p\}, y_k \leq u_k$.*
- *A lower bound on Y_N is a point $l = (l_1, \ldots, l_p) \in \mathbb{R}^p$ such that for all $y \in Y_N$, for all $k \in \{1, \ldots, p\}, l_k \leq y_k$.*

Definition 9 (Ideal and nadir points)**.**

- *The ideal point is y^I such that $y^I_k = \max\{y_k : y \in Y\}, k = 1, \ldots, p$.*
- *The nadir point is y^N such that $y^N_k = \min\{y_k : y \in Y_N\}, k = 1, \ldots, p$.*

The ideal point y^I and nadir point y^N (Definition 9) are commonly used as upper and lower bounds for Y_N.

Based on Definition 9, it is obvious that y^I dominates every nondominated points and y^N is dominated by every nondominated points. y^I and y^N are respectively the tightest upper and lower bound for Y_N. However, Ehrgott and Gandibleux (2001) remarks three main drawbacks to the use of y^I and y^N as bounds for a MOCO problem:

- y^I can be difficult to find if the single-objective version of the problem is $\mathcal{NP}$-hard, since it involves to solve p $\mathcal{NP}$-hard problems.
- y^N is difficult to obtain when $p > 2$; Ehrgott and Tenfelde-Podehl (2003) and Kirlik and Sayın (2014) remark this even if the single-objective version of the problem is in $\mathcal{P}$.
- y^I and y^N are generally located far away from the nondominated points.

In order to obtain a finer estimation of the nondominated points, the notion of bound has been extended to bound sets, considering a set of points for bounding Y_N.

Several definitions of bound sets have been proposed, cf (Villarreal and Karwan, 1981), (Ehrgott and Gandibleux, 2001), (Ehrgott and Gandibleux, 2007).

Villarreal and Karwan (1981) were the first to attempt to give a general formal definition of bound sets for Y_N, which they call sets of bounds. The definition of those sets of bounds (see Definition 10) is based on a pairwise comparison of the nondominated points with the points of the set of bounds.

Definition 10 (Bound sets (Villarreal and Karwan, 1981))**.** *A set of upper bounds U for the solution of a MOILP problem P is a set of points that satisfies the following conditions:*

- *Each point of U is either a nondominated point or dominates at least one of the nondominated point of P.*
- *Each nondominated point of P is dominated by at least one member of U or is a member of U.*

A set of lower bounds L for the set of nondominated points for a MOILP problem P is a set of points that satisfies the following condition:

- *Each element of L is either a nondominated point or is dominated by at least one nondominated point of P.*

In (Ehrgott and Gandibleux, 2001) the authors give a different definition of bound sets.

Definition 11 (Bound sets (Ehrgott and Gandibleux, 2001))**.** *Let $\bar{X}$ be a subset of X_E and $\bar{Y} = z(\bar{X})$.*

An upper bound set for Y_N is a subset $U \subset \mathbb{R}^p_{\geq}$ such that:

- *For each $y \in \bar{Y}$ there is some $u = (u^1, \ldots, u^p) \in U$ such that $u_k \geq y_k, k = 1, \ldots, p$.*
- *There is no pair $y \in \bar{Y}, u \in U$ such that y dominates u.*

A lower bound set for $\bar{Y}$ is a subset $L \subset \mathbb{R}^p_{\geq}$ such that:

- *For each $y \in \bar{Y}$ there is some $l = (l^1, \ldots, l^p) \in L$ such that $y_k \geq l_k, k = 1, \ldots, n$.*
- *There is no pair $y \in \bar{Y}, l \in L$ such that l dominates y.*

Concerning the upper bound set, Definition 11 is more general than Definition 10:

- In (Ehrgott and Gandibleux, 2001), an upper bound set is defined for any subset of Y_N whereas it was only defined for Y_N in (Villarreal and Karwan, 1981). This remark also holds for the definitions of lower bound sets.
- Definition 10 stipulates that every point in the upper bound set U must dominate at least one point of Y_N. This restriction does not exist in Definition 11 (we consider that $\bar{Y} = Y_N$), thus the upper bound set may contain points that do not dominate any $y \in Y_N$ as long as they

are not dominated by any point of $y \in Y_N$ and every $y \in Y_N$ is dominated by at least one point of U. As a consequence, $[\text{conv}(Y_N - \mathbb{R}^p_{\geqq})]_N$ is an upper bound set for Y_N according to Definition 11, whereas it may not satisfy Definition 10 (see Figure 1.3).

The definition of lower bound set also differ in Definitions 10 and 11. Indeed the later requires that for every $y \in Y_N$, there exists $l \in L$ such that l is weakly dominated by y (with $\bar{Y} = Y_N$); while the former only requires that every $l \in L$ is weakly dominated by a $y \in Y_N$. Thus Y_{SN} is lower bound set according to Definition 10 but not necessarily according to Definition 11. For the latter, the local nadir points of Y_{SN} define a lower bound set.

In this work we use the definition proposed in (Ehrgott and Gandibleux, 2007), in which the definitions of upper and lower bound sets are not symmetrical. Before introducing their definition, we need some additional terminology. We consider $S \subset \mathbb{R}^p$:

- S is $\mathbb{R}^p_{\geqq}$-closed if the set $S - \mathbb{R}^p_{\geqq}$ is closed.
- S is $\mathbb{R}^p_{\geqq}$-bounded if there exists $s^0 \in \mathbb{R}^p$ such that $S \subset s^0 - \mathbb{R}^p_{\geqq}$.
- cl S is the closure of S.
- S^c is the complement of S.

Definition 12 (Bound sets (Ehrgott and Gandibleux, 2007))**.** *Let $\bar{Y} \subset Y_N$.*

An upper bound set U for $\bar{Y}$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set $U \subset \mathbb{R}^p$ such that $\bar{Y} \subset U - \mathbb{R}^p_{\geqq}$ and $U \subset (U - \mathbb{R}^p_{\geqq})_N$.

A lower bound set L for $\bar{Y}$ is an $\mathbb{R}^p_{\geqq}$-closed and $\mathbb{R}^p_{\geqq}$-bounded set $L \subset \mathbb{R}^p_{\geqq}$ such that $\bar{Y} \in \text{cl}[(L - \mathbb{R}^p_{\geqq})^c]$ and $L \subset (L - \mathbb{R}^p_{\geqq})_N$.

In the definition, $L \subset (L - \mathbb{R}^p_{\geqq})_N$ (respectively $U \subset (U - \mathbb{R}^p_{\geqq})_N$) specifies that the points of L (respectively U) cannot dominate each other.

$\bar{Y} \in \text{cl}[(L - \mathbb{R}^p_{\geqq})^c]$ stipulates that there cannot exist a point in $\bar{Y}$ dominated by a point in L.

$\bar{Y} \subset U - \mathbb{R}^p_{\geqq}$ imposes that no point of $\bar{Y}$ can dominate a point of U, i.e. that the upper bound set U is "above" $\bar{Y}$.

Definitions 11 and 12 differ for the lower bound sets. Indeed in the later, it is not necessary that every $y \in \bar{Y}$ is dominated by at least one point of L. Hence, Definition 12 accepts as lower bound set any subset of feasible solutions, filtered by dominance, as in Definition 10, whereas Definition 11 does not.

We can remark that y^I and y^N are bound sets according to the three definitions 10, 11 and 12.

In this manuscript, when the set of points to bound $\bar{Y}$ is not specified we will consider that the bounds or bound sets are defined for Y_N. By abuse of language, we call a *convex upper bound set* U for $\bar{Y}$ an upper bound set for $\bar{Y}$ such that $U - \mathbb{R}^p_{\geqq}$ is convex.

Figure 1.3 illustrates Definitions 10, 11 and 12 and y^I and y^N from Definition 9.

In single-objective optimization, two upper bounds for Y_N are always comparable, since they are scalars. This is not the case in the multi-objective context. In (Ehrgott and Gandibleux, 2007), the definition of dominance between upper bound sets is formulated as follows:

Definition 13 (Dominance of upper bound sets (Ehrgott and Gandibleux, 2007))**.** *Given two upper bound sets U^1 and U^2 for a same set $\bar{Y}$, U^1 dominates U^2 if $U^1 \subset U^2 - \mathbb{R}^p_{\geqq}$ and $U^1 - \mathbb{R}^p_{\geqq} \neq U^2 - \mathbb{R}^p_{\geqq}$.*

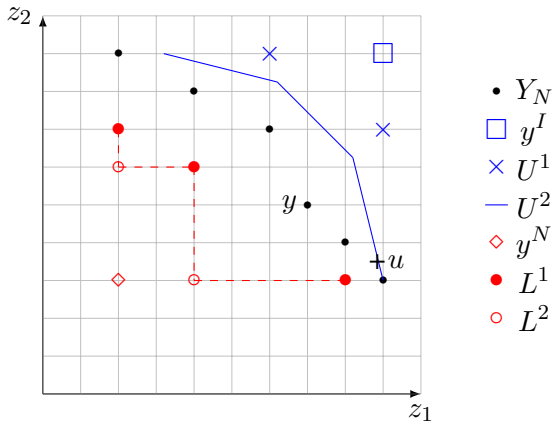
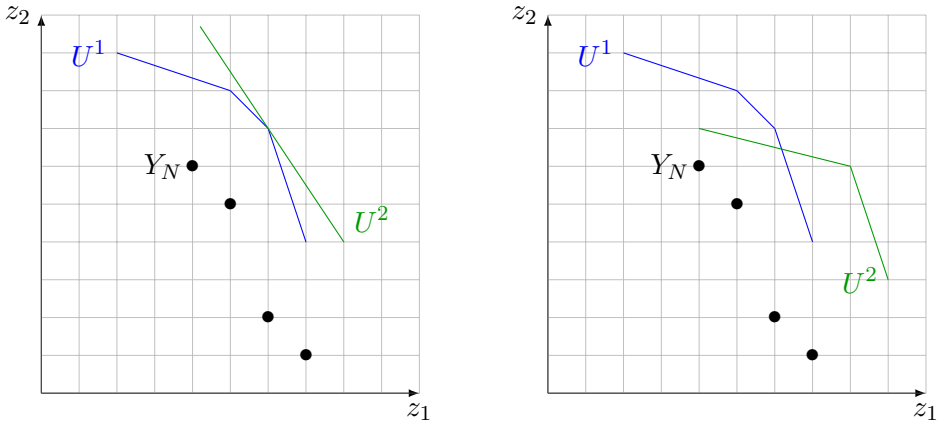


Figure 1.3: y^I and y^N are an upper and a lower bound for Y_N.
U^2 is an upper bound set for Y_N according to Ehrgott and Gandibleux (2001) and Ehrgott and Gandibleux (2007), but not according to Villarreal and Karwan (1981) since u (depicted by $+$) does not dominate any $y \in Y_N$. U^1 is an upper bound set for Y_N according to Definitions 10, 11 and 12.
L^1 is a lower bound set for Y_N according to Villarreal and Karwan (1981) and Ehrgott and Gandibleux (2007), but not according to Ehrgott and Gandibleux (2001) since y does not dominate any point of L^1. L^2 is a lower bound set for Y_N according to Definitions 10, 11 and 12.

This definition establishes that the upper bound set U^1 dominates U^2 if U^1 is "below" U^2. Obtaining this relation between upper bound sets is rare. Indeed generally when comparing two upper bound sets U^1 and U^2, there exists $u \in U^1$ such that $u \in U^2 - \mathbb{R}^p_{\geq}$ and there exists $u \in U^2$ such that $u \in U^1 - \mathbb{R}^p_{\geq}$ (there are some parts of U^1 which are "below" U^2 and some part of U^2 which are "below" U^1). Then we say that U^1 and U^2 are incomparable.

Figure 1.4 gives a graphical example of an upper bound set for Y_N dominating another and of two incomparable upper bound sets for Y_N.



(a) U^1 dominates U^2 (b) U^1 and U^2 are incomparable

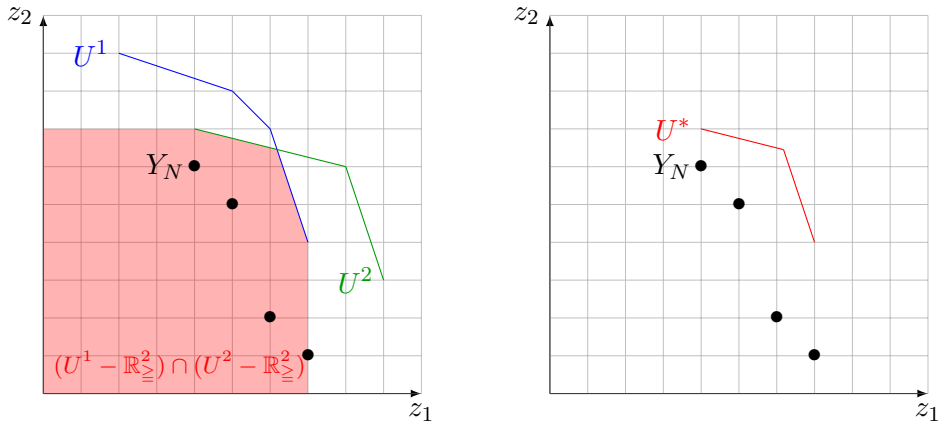
Figure 1.4: Comparison of two upper bound sets U^1 and U^2 for Y_N

In Proposition 1, from (Ehrgott and Gandibleux, 2007), two upper bound sets for $\bar{Y}$ are "merged" in order to create a third upper bound set for $\bar{Y}$ which dominates both of the former

upper bound sets.

Proposition 1 ((Ehrgott and Gandibleux, 2007)). *If U^1, U^2 are upper bound sets for $\bar{Y}$ and $U^1 - \mathbb{R}^p_{\geqq} \neq U^2 - \mathbb{R}^p_{\geqq}$ then $U^* = [(U^1 - \mathbb{R}^p_{\geqq}) \cap (U^2 - \mathbb{R}^p_{\geqq})]_N$ is an upper bound set dominating U^1 and U^2.*

Figure 1.5 illustrates Proposition 1.



(a) U^1 and U^2 are two upper bound sets for Y_N

(b) U^* defined by Proposition 1 dominates U^1 and U^2

Figure 1.5: Illustration of Proposition 1

1.3 Relaxations

Bounds or bound sets are often a key component of exact solution methods, for single-objective as well as for multi-objective problems. Relaxations are used to compute upper bounds or bound sets in exact methods. A relaxation of a problem is a simplification of the initial problem and is defined, for the single-objective case, in Definition 14.

Definition 14 (Relaxation in the single-objective context (Wolsey, 1998)). *A problem (RP) $z^R = \max\{f(x) : x \in X' \subseteq \mathbb{R}^n\}$ is a relaxation of a single-objective problem (P) $z^* = \max\{z(x) : x \in X \subseteq \mathbb{R}^n\}$ if:*

- *$X \subseteq X'$*
- *$f(x) \geq z(x)$ for all $x \in X$, $f : \mathbb{R}^n \to \mathbb{R}$ and $z : \mathbb{R}^n \to \mathbb{R}$*

The main property of a relaxation is that $z^R \geq z^*$.

The direct extension of this definition to the multi-objective context involves $f : \mathbb{R}^n \to \mathbb{R}^p$ and $z : \mathbb{R}^n \to \mathbb{R}^p$ and the second item becomes: $f(x) \geqq z(x)$ for all $x \in X$.

In this section, we only present the three relaxations that we will use in this manuscript. However, there exists several other ones, such as the Lagrangian relaxation, the composite relaxation, the combinatorial relaxation, etc.

1.3.1 Linear programming relaxation

In the linear programming relaxation (LP relaxation) of an optimization problem with integer variables, the constraints on the integrality on the value of the variables are omitted. For an ILP,

let suppose that $x_j \in \{m, \dots, M\}$, then this constraint is replaced by $m \leq x_j \leq M$ in the LP relaxation, $j \in \{1, \dots, n\}$. For example, for a problem with binary variables, the constraints $x_j \in \{0, 1\}$ are replaced by $0 \leq x_j \leq 1$ for $j = 1, \dots, n$. The objective functions are unchanged. For a single-objective optimization problem, we denote z^* the optimal value of the problem and z^{LP} the optimal solution of its LP relaxation. Obviously we have $z^{LP} \geq z^*$.

The extension of the LP relaxation to a multi-objective problem is straightforward. The relaxed problem being a MOLP, there generally exists an infinite number of efficient solutions. This problem can be solved by using a dichotomic method or a multi-criteria simplex. The nondominated points of this relaxed problem constitute a convex upper bound set for Y_N of the original problem.

1.3.2 Convex relaxation

The convex relaxation of a problem consists in relaxing its feasible set X, by considering as feasible the convex hull of X. Since it does not affect the objective functions, its definition is similar for single-objective and multi-objective contexts. However, solving the relaxed problem differs depending on the number of considered objective function.

For a single-objective problem, the optimal solution of its convex relaxation is the optimal solution of the original problem. Finding the formulation of $\operatorname{conv} X$ is the purpose of polyhedral methods. Finding the ideal linear formulation of $\operatorname{conv} X$ is generally an $\mathcal{NP}$-hard problem (Wolsey, 1998).

In the multi-objective context, the reformulation of the feasible set is not required to solve the convex relaxation. Indeed, the nondominated points of the convex relaxation is $(\operatorname{conv} Y_N)_N$ for the original problem. $(\operatorname{conv} Y_N)_N$ is an upper bound set for Y_N and is even the tightest convex upper bound set for Y_N, i.e. it dominates any other upper bound set for Y_N. The extreme points of $(\operatorname{conv} Y_N)_N$ are the supported nondominated points of the problem. They can be found using a dichotomic method (see Section 1.4.1) using the weighted sum problem. The convex relaxation of MOILP, MOMILP and MOCO problems are easy to solve if the single-objective version of the problem can be solved by a polynomial or pseudo-polynomial time complexity algorithm.

1.3.3 Surrogate relaxation

The surrogate relaxation was introduced for the first time in (Glover, 1965), for single-objective problems. It aggregates l constraints by summing them, using a multiplier $\mu \in \mathbb{R}^l_{\geqq}$. For an ILP problem P, the surrogate relaxed problem $S(\mu)$ is the following:

$$
\begin{array}{lll}
\max & z(x) & \\
\text{s.t.} & \displaystyle\sum_{i=1}^{l} \mu_i \, A_i \, x \leq \sum_{i=1}^{l} \mu_i \, b_i & \\
& A_i \, x \geqq b_i & \forall i \in \{l+1, \dots, m\} \\
& x \in \mathbb{Z}^n &
\end{array}
\qquad (SR(\mu))
$$

where A_i is the i-th line of the constraint matrix A.

All the feasible solutions for P are feasible for $SR(\mu)$ with $\mu \in \mathbb{R}^l_{\geqq}$. Thus the value of the optimal solution of $SR(\mu)$ is an upper bound for the feasible solutions of P. The value of the upper bound depends on the multiplier μ used. The surrogate dual problem SD provides the tightest upper bound (Glover, 1975).

$$z(SD) = \min_{\mu \in \mathbb{R}^p_{\geqq}} z(SR(\mu))$$

The extension of the surrogate relaxation to a multi-objective context is straightforward. Indeed, the surrogate relaxation deals only with the decision space, the multiplicity of the objective functions does not impact the definition of the surrogate relaxation. However, the extension of the surrogate dual problem is more difficult. Indeed the surrogate relaxation being a MOCO problem, its nondominated set is a discrete. Two upper bound sets obtained by two different surrogate relaxation, using different multipliers, are not necessarily comparable.

1.4 Solution methods for multi-objective combinatorial optimization problems

In this section we describe different solution methods for MOCO problems. Reviews of the literature on MOCO problems can be found in (Ehrgott and Gandibleux, 2000) and (Ehrgott and Gandibleux, 2002). They present applications of MOCO problems and present different dedicated solution methods.

Even if Section 1.4.7 deals with approximation solution methods, we mainly present exact solution methods, since the thesis focuses on the exact solution of knapsack problems. We particularly detail branch-and-bound and two phase methods. For most of these methods, we highlight the difference between the bi-objective version and the version with strictly more than two objectives. Indeed, bi-objective solution methods often rely on the natural order of the nondominated points.

1.4.1 Dichotomic method

This first method does not aim to find X_{E_m}, but guarantee to find X_{SE1_m}. Some other solutions in X_{SE1} or X_{SE2} can potentially be found. For a problem P, the dichotomic method is based on its weighted sum scalarization P_λ defined by Geoffrion (1968).

Aneja and Nair (1979) first introduced this method for bi-objective transportation problems. The pseudo-code of the algorithm is presented in Algorithm 1.

The initialization of the method (lines 2 to 10) consists in finding the two lexicographic optimal solutions x^1 and x^2, for the permutation of objective functions (1,2) and (2,1) respectively. When searching for the lexicographic optimal solutions, to avoid weakly efficient solutions that are not efficient solutions, ε is used in the directions. Thus ε should be a small value.

At lines 11 to 18, the method searches iteratively for the other supported efficient solutions by successively considering each pair of adjacent (regarding the natural order) supported nondominated points found during the execution. During the first iteration, the only pair is $z(x^1), z(x^2)$. Each iteration consists in computing a search direction, searching the optimal solution x^* in this direction and updating the pair of adjacent supported nondominated points. For a pair of points y^r, y^l, such that $y^l_1 < y^r_1$ (and $y^l_2 > y^r_2$ due to the natural order), the direction computed according to the formula of line 13 is orthogonal to the line (y^r, y^l). The next steps of the execution depend on the point $z(x^*)$ obtained during the solution. If $z(x^*)$ is above the line (y^r, y^l), i.e. if $\lambda\, z(x^*) > \lambda\, y^r$ ($\lambda\, y^l = \lambda\, y^r$), then two new pairs have to be considered: $y^r, z(x^*)$ and $z(x^*), y^l$. If this is not the case, the search stops in for this direction.

Lines 6 and 7 correspond to the specific case of a unique nondominated point for the problem P.

Figure 1.6 illustrates graphically an execution of the dichotomic method. The arrows represent the optimization direction used for each iteration.

Algorithm 1: Dichotomic method (Aneja and Nair, 1979)

input : A bi-objective combinatorial optimization problem instance
output: The X_{SE1_m} of this instance

begin
  /* getOptimalP($\lambda\downarrow$) returns an optimal solution of P_λ */
  /* addToS($y^r\downarrow, y^l\downarrow$) adds the pair y^r, y^l to the set of pairs S */
  /* popFromS() returns the first pair y^r, y^l of S and deletes it from S */
  /* $\varepsilon > 0$ is a small value */
  $X_{SE_m} \leftarrow \emptyset$
  $S \leftarrow \emptyset$
  $x^1 \leftarrow$ getOptimalP($(1, \varepsilon)\downarrow$)
  $x^2 \leftarrow$ getOptimalP($(\varepsilon, 1)\downarrow$)
  if $z(x^1) = z(x^2)$ **then**
    $X_{SE1_m} \leftarrow \{x^1\}$
  else
    addToS($z(x^1)\downarrow, z(x^2)\downarrow$)
    $X_{SE1_m} \leftarrow \{x^1, x^2\}$
    while $S \neq \emptyset$ **do**
      $y^r, y^l \leftarrow$ popFromS()
      $\lambda \leftarrow (y^l_2 - y^r_2, y^r_1 - y^l_1)$
      $x^* \leftarrow$ getOptimalP($\lambda\downarrow$)
      if $\lambda\, z(x^*) > \lambda\, y^r$ **then**
        $X_{SE1_m} \leftarrow X_{SE_m} \cup \{x^*\}$
        addToS($y^r\downarrow, z(x^*)\downarrow$)
        addToS($z(x^*)\downarrow, y^r\downarrow$)
  return X_{SE1_m}

Comment. In the algorithms, the symbols $\downarrow$, $\uparrow$ and $\updownarrow$ specify the transmission mode of a parameter to a procedure; they correspond respectively to the mode IN, OUT and IN OUT.

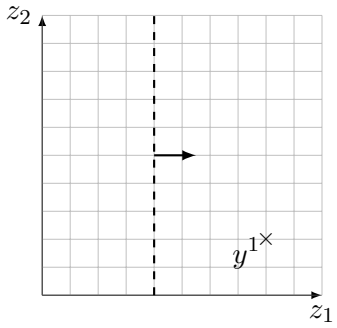
We can remark that this method is also applicable to MOLP and MOILP problems, by adjusting the function *getOptimal*.

The dichotomic method has been adapted for strictly more than two objectives in (Przybylski et al., 2010b) and (Özpeynirci and Köksalan, 2010). In this context, the search directions are composed of three or more dimensions and can be defined by more than two points. Thus determining the directions to find all supported nondominated point is a more complex process. Since the scope of our work does not include problems with more than two objective functions, we do not develop those methods in this manuscript.

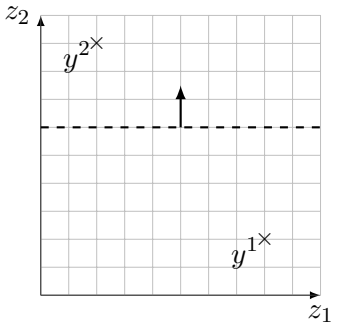
Obviously, the dichotomic method does not allow to find non-supported efficient solutions and does not guarantee to find all non-extreme supported efficient solutions. Section 1.4.6 shows how the dichotomic method can be combined with a enumeration method to find X_{E_m}.

1.4.2 ε-constraint method

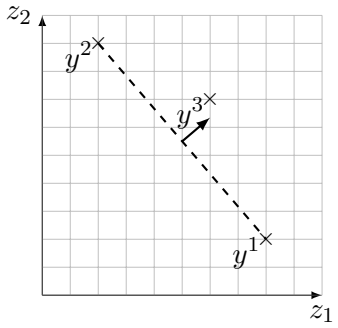
The ε-constraint method was first introduced in (Haimes et al., 1971); it allows to find X_{E_m}. Contrary to the dichotomic method, this method does not aggregate the constraints. It transforms



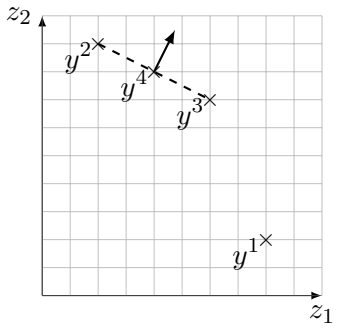
(a) Initialization: search of the lexicographic optimal for the permutation (1,2). y^1 is obtained.



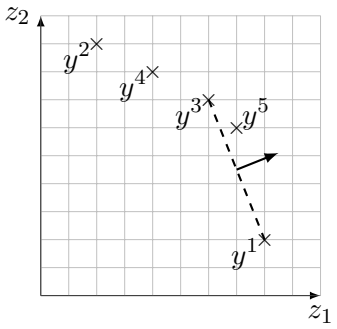
(b) Initialization: search of the lexicographic optimal for the permutation (2,1). y^2 is obtained.



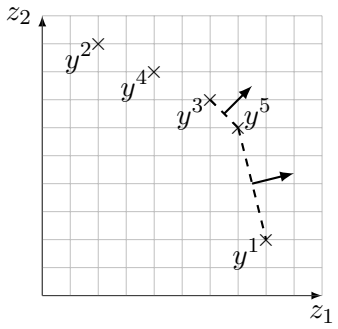
(c) Iteration 1: the search direction is defined by y^2, y^1. y^3 is obtained with $\lambda y^3 > \lambda y^2$.



(d) Iteration 2: the search direction is defined by y^2, y^3. y^4 is obtained, with $\lambda y^4 = \lambda y^2$



(e) Iteration 3: the search direction is defined by y^3, y^1. y^5 is obtained with $\lambda y^5 > \lambda y^3$



(f) Iteration 4 and 5: the search direction is defined by y^3, y^5 and y^5, y^1. No new point is obtained.

Figure 1.6: Illustration of the dichotomic method

the multi-objective problem into a single-objective one by keeping only one of the original objective functions, the other objective functions are transformed into constraints. The resulting problem is called the ε-constraint problem:

$$\begin{array}{ll} \max & z_l(x) \\ \text{s.t.} & z_k(x) \geq \varepsilon_k \, \forall k \in \{1, \ldots, p\} \setminus \{l\} \\ & x \in X \end{array}$$

where $\varepsilon \in \mathbb{R}^p_{\leqq}$. Solving this problem, we obtain at least a weakly efficient solution.

We present the ε-constraint method in a MOCO context with $p = 2$.

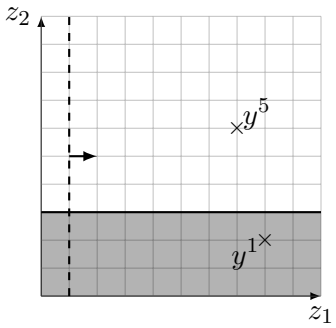
To find X_{E_m}, the ε-method iteratively solves the ε-problem using different values for ε. The first iteration aims to find the optimal solution of the problem by considering only the first objective function. There is no constraint on the other objective. At the $(i+1)$-th iteration, the vector ε is chosen such that the weakly efficient solutions found during the first i iterations are not feasible for the ε-constraint problem, while all the other efficient solutions remain feasible. Let us consider x^i, the optimal solution of the i-th iteration. The following ε-constraint problem is the problem used for the $(i+1)$-th iteration:

$$\begin{array}{ll} \max & z_1(x) \\ \text{s.t.} & z_2(x) \geq z_2(x^i) + \epsilon \\ & x \in X \end{array}$$

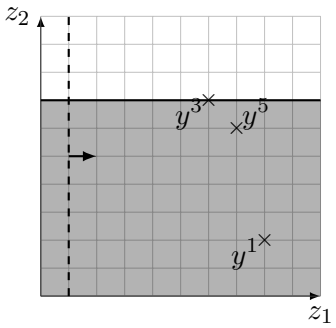
When dealing with some non-integer coefficients for the objective function, the value ϵ has to be a small value, guaranteeing that there is no efficient solutions x with $z_2(x) \in \left]z_2(x^i), z_2(x^i) + \epsilon\right[$.

When the coefficients of the objective function are integer, ϵ can be set to 1, thanks to the integrity of the variables. Indeed there is no possible value for the second objective function in the open interval $\left]z_2(x^i), z_2(x^i) + 1\right[$.

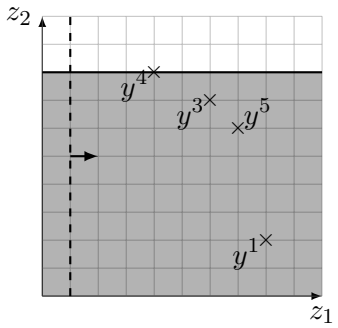
Figure 1.7 is an illustration of the ε-constraint method on a bi-objective problem. At each iteration, the plain line indicates the ε-constraint introduced, the shadowed area corresponds to the non-feasible area with regards to the ε-constraint and the arrow indicates the optimization direction. The first iteration is not presented since it is similar to Figure 1.6a (the second objective is not constrained).



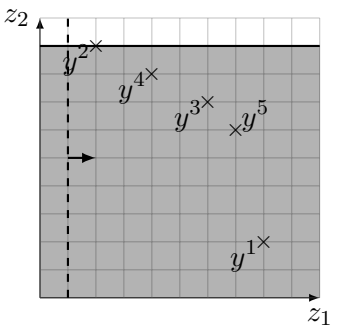
(a) Iteration 2. y^1 is the lexicographic optimal solution for the permutation (1,2). y^5 is obtained



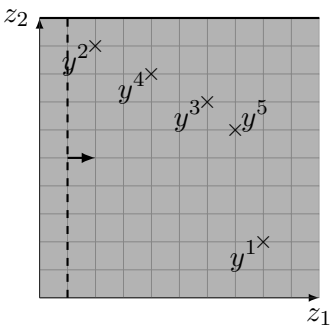
(b) Iteration 3. y^3 is obtained



(c) Iteration 4. y^4 is obtained



(d) Iteration 5. y^2 is obtained



(e) Iteration 5. No solution is obtained

Figure 1.7: Illustration of the ε-constraint method

Generalization of the ε-constraint method for optimization problems with three objective functions or more have been proposed by Laumanns et al. (2005), Lokman and Köksalan (2013), Özlen et al. (2014), Kirlik and Sayın (2014) and Dächert and Klamroth (2014). In the latter, the algorithm requires the solving of a linear number of subproblems.

1.4.3 Branch-and-bound method

The branch-and-bound method is an implicit enumeration method to find X_{E_m}. The feasible set is successively divided by fixing one or several variables. The partitioning of the feasible set can be represented by a tree whose root node is the global problem (i.e. the original problem to solve) and each node represents a subproblem, in which certain variables are fixed. The variables of a subproblem which are not fixed are called *free variables*.

In order to perform an implicit enumeration of all solutions, each subproblem is evaluated. This evaluation aims to prove if no efficient solution for the global problem is feasible for this subproblem. In this case the node is fathomed, i.e. the partitioning stops in this branch. In the following, we denote $\bar{Y}_N$ the nondominated points of the subproblem associated to a node and Y_N the nondominated points of the global problem.

The branch-and-bound method was introduced for single-objective optimization problems before being generalized to multi-objective problems. Kiziltan and Yucaoğlu (1983) presents the first generalization to multi-objective integer optimization, with binary variables. This section only deals with multi-objective branch-and-bound for binary problems. Most of the multi-objective branch-and-bound methods are dedicated to a particular problem: the bi-objective minimum weight spanning tree problem for (Ramos et al., 1998) and (Sourd and Spanjaard, 2008), the bi-objective assignment problem for (Delort and Spanjaard, 2010). The knapsack problems are the focus of some multi-objective branch-and-bound methods (see Section 2 for details). Vincent et al. (2013) has developed a branch-and-bound method for MOMILP problems. All works will be presented for maximization problems, with non-negative coefficients and variables.

We describe the different strategic stages of a branch-and-bound method.

Separation procedure

The separation procedure partitions the feasible set X. Generally, for binary variables, the partitioning procedure consists in fixing one variable to 0 or 1. The choice of the variable to fix during the partitioning procedure is called the branching strategy. The branching strategy can be either static (fixing variables in a pre-established order) or dynamic (using information of the enumeration to make the choice).

Static strategies are more often used, particularly in multi-objective contexts. The static branching strategies are most of the time problem dependent.

Lower bound set

In a branch-and-bound method, a lower bound set on Y_N aims to ease the pruning of branches. The branch-and-bound method keeps an updated list of feasible solutions found during the enumeration, filtered by dominance. A solution in this list is called a *potentially efficient solution*, since it is a feasible solution such that no solution dominating it have been found, at a given stage of the execution, even if this could happen later in the execution. The points associated to those solutions constitute the lower bound set and are called *potentially nondominated points*. At the beginning of the algorithm, if no feasible solution is available, the lower bound set is initialized with the point $(0, \ldots, 0)$. To improve its quality an initialization can be performed (see paragraph *Obtaining new feasible solutions*).

Upper bound set

At each node, an upper bound set U for $\bar{Y}_N$ (the nondominated set of the subproblem) is computed, in order to evaluate if the node can be fathomed. Generally the upper bound set is obtained thanks to a relaxation of the subproblem. This strategic stage has evolved from using an upper bound (reduced to a single point) to using an upper bound set (defined by several points).

In the earliest multi-objective branch-and-bound methods, the upper bound is often an utopia point, i.e. a point $y^U = (y_1^I + \epsilon_1, \ldots, y_p^I + \epsilon_p)$ where $\epsilon_k > 0, k = 1, \ldots, p$. For example, in (Kiziltan and Yucaoğlu, 1983), the upper bound is the ideal point of the unconstrained version of the subproblem. In (Ulungu and Teghem, 1997) and (Florios et al., 2010), the upper bound is also an utopia point.

In (Visée et al., 1998) the authors use three upper bounds: two for the problems considering only one objective function (they are in a bi-objective context) and one for a weighted sum problem. This method will be detailed in Section 2.4.

In more recent works, upper bound sets are used to evaluate the subproblem. Mostly convex upper bound sets are considered. The convex relaxation is used to compute the upper bound set in many recent works, as in (Sourd and Spanjaard, 2008), (Jorge, 2010) and (Delort, 2011)). As mentioned in Section 1.2.4, the convex relaxation leads to a particularly tight convex upper bound set and is easy to compute when the single-objective version is solvable in polynomial or pseudo-polynomial time complexity. The use of the convex relaxation to compute the upper bound set appears to be an important factor of the practical efficiency of recent multi-objective branch-and-bound methods.

Fathoming of nodes

A node is fathomed when it is proven that the feasible solutions for the associated subproblems cannot be efficient solutions of the global problem. We can distinguish three ways for a node to be fathomed:

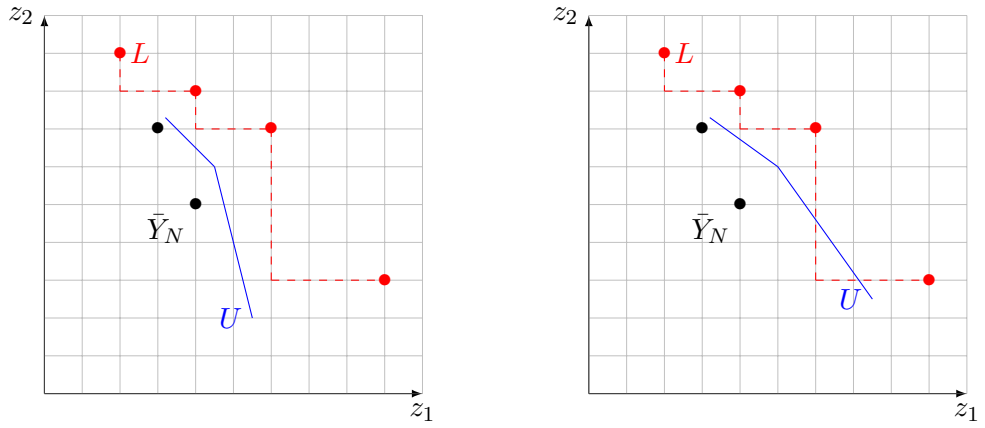
By infeasiblity If the subproblem defined does not have any feasible solution, then it is obvious that no efficient solution for the global problem can be feasible for this subproblem. We can remark that the feasibility of a subproblem only depends on the constraints, and not on the number of objectives.

By optimality If the solutions associated to the upper bound set are feasible, then they are efficient for the subproblem. But it does not guarantee that all efficient solutions of the subproblems have been found. To guarantee that, we would have to determine a lower bound set for $\bar{Y}_N$ with the equality $U - \mathbb{R}^p_{\geqq} = L - \mathbb{R}^p_{\geqq}$ with U the upper bound set on $\bar{Y}_N$ and L the lower bound set on $\bar{Y}_N$, which is unlikely to happen in the multi-objective context. Indeed, the upper bound and the lower bound sets are generally of different nature (convexity, bound or bound set). However, there exists one special case: if the upper bound set is composed of a single point which is feasible for the subproblem, then the node can be fathomed by optimality.

By dominance Fathoming a node by dominance requires to compare the upper bound set U computed for $\bar{Y}_N$ (subproblem) and the lower bound set L for Y_N (global problem). The node can be fathomed if U is dominated by L, i.e. for all $u \in U$ there is $l \in L$ such that $l \geqq u$ (see Figure 1.8a). Then obviously all feasible solutions of the subproblem are weakly dominated by a feasible point in L and no new nondominated points for the global problem can be found in this branch of the search tree.

However, if a single point of U is not weakly dominated by any point of L then the node cannot be fathomed (see Figure 1.8b).

The fathoming by dominance emphases the importance of the quality of the upper and lower bound sets. Indeed when using a tight upper bound set and a lower bound set composed of feasible solutions of good quality then the nodes are more likely to be fathomed by dominance.



(a) U is dominated by L. The node can be fathomed

(b) U is not dominated by L. The node cannot be fathomed

Figure 1.8: Comparison of an upper bound set U for $\bar{Y}_N$ with a lower bound set L for Y_N to determined if the node can be fathomed

Moreover, when dealing with integer or binary variables and coefficients, the integrity can be used. Let us consider L the lower bound set for Y_N and U the upper bound set for $\bar{Y}_N$ such that $(L + \mathbb{R}^p_{\geqq}) \cap (U - \mathbb{R}^p_{\geqq}) \neq \emptyset$ but does not contain any integer point. Then no new nondominated point can be found in this branch. However, the fathoming rule by dominance must be strengthened. In (Sourd and Spanjaard, 2008), the authors use this integrality and remark that each point of the lower bound set for Y_N can be shifted by (1,1) without losing any nondominated points. This shifted lower bound set, illustrated in Figure 1.9, makes it possible to prune the node in the situation above.

Choice of the active node

An active node is a node that has not been fathomed, i.e. corresponding to a subproblem for which the feasible solutions may be efficient solutions of the global problem. Those nodes must be investigated. The branch-and-bound method investigates the nodes iteratively by selecting at each step one node in order to apply the separation procedure. It is well known that the order in which the nodes are investigated influences the size of the search tree. Indeed, when investigating a node, some new potentially nondominated points can be found, improving the lower bound set for Y_N, which can make it possible to fathom by dominance some active nodes.

The choice of the active node can either be static (depth first search or breadth first search) or dynamic (for example, choosing the node with the highest value for the upper bound, called *best value first* strategy in single-objective optimization). Most of the multi-objective branch-and-bound methods use the depth first search strategy. Even if the best value first strategy is used in the single-objective context (the bounds are always comparable), its definition is complex in

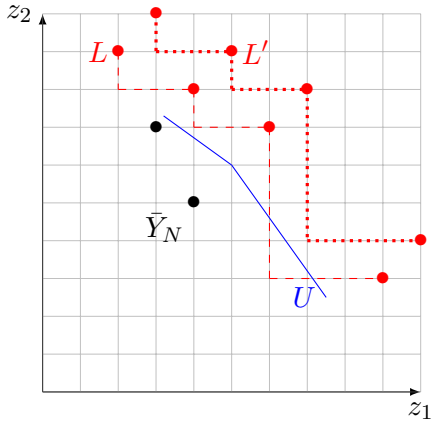


Figure 1.9: The upper bound set U for $\bar{Y}_N$ is not dominated by the lower bound set L on Y_N, however by using the shifted lower bound set L' for Y_N the node can be fathomed.

the multi-objective context because dominance relation cannot always be determined between two upper bound sets. A measure of the quality of an upper bound set can be defined in several ways, as in (Jorge, 2010) to choose the active node.

Generating new feasible solutions

Generating new feasible solutions is important for two reasons. Firstly, it makes it possible to find the solutions of a set X_{E_m}, which have to be returned by the algorithm. Secondly, the quality of the lower bound set is essential for the practical efficiency of the method, facilitating the fathoming of nodes by dominance during the execution.

New feasible solutions can be found either at the beginning of the algorithm, initializing the lower bound set, and/or all along the search-tree. The quality of the feasible solutions found is important for the practical efficiency of the method.

Usually the initialization of the lower bound set is achieved by computing Y_{SN} (by solving the convex relaxation of the global problem) as in (Ramos et al., 1998), (Sourd and Spanjaard, 2008), (Jorge, 2010) or (Delort and Spanjaard, 2010); or by using a heuristic (see (Sourd and Spanjaard, 2008) or (Delort and Spanjaard, 2010)). We can note that it is possible to apply both procedures or to execute a metaheuristic to find efficient solutions of good quality.

In (Florios et al., 2010), feasible solutions are generated during the initialization. The LP relaxation of the problem is solved. The extreme efficient solutions of this relaxation are generally not integral and thus not feasible for the original problem. The authors round the variables, aiming to generate feasible solutions.

Feasible solutions can also be found all along the algorithm (it is actually essential to return X_{E_m}). The most basic method to find a feasible solution is to develop a branch of the search-tree until the subproblem does not contain any free variable, i.e. at a leaf of the search-tree. Then the corresponding solution can be used to improve the lower bound set, if it is feasible and if its image in objective space is not dominated by any point of the lower bound set. Even if this process can be used when the choice of the active node is a depth first search, it has the drawback that the branch has to be entirely developed, favoring large search-trees.

Finding feasible solutions before reaching a leaf node can ease the fathoming of nodes by dominance and thus reduce the search-tree. Heuristics can be used to generate feasible solutions. For example, for a knapsack problem, the solution for which all free variable are fixed to 0 is feasible. The algorithms presented in (Ulungu and Teghem, 1997) and (Visée et al., 1998) generate this solution at each node. In an other context, this solution is not necessarily feasible. However, Kiziltan and Yucaoğlu (1983) generates this solution and tests its feasibility at each node, in order to improve the lower bound set.

The generation of new feasible solutions during the enumeration is often linked to the computation of the upper bound set. Indeed, when the upper bound set contains feasible solutions, those solutions might improve the lower bound set. For example, when using the convex relaxation, see (Sourd and Spanjaard, 2008), (Jorge, 2010) and (Delort, 2011), the extreme solutions are feasible for the subproblem and the global problem. Thus they are likely to improve the lower bound set for Y_N.

1.4.4 Branch-and-cut

The branch-and-cut method is derived from the branch-and-bound method. The branch-and-cut method has been originally defined for single-objective optimization problems (Wolsey, 1998). During the search process, the upper bound is tightened by introducing valid inequalities, i.e. additional constraints that do not modify the feasible set but which can be violated by solutions of a relaxation (for example by the optimal solution of the LP relaxation). Wolsey (1998) stated that the philosophy of a branch-and-cut method is not the same as for a branch-and-bound method, since a particular effort is spent on tightening the upper bound. To obtain an efficient branch-and-cut method, one must find the tradeoff between the quality of the upper bound obtained at each node and the computational time spent on the generation of cuts. The term branch-and-cut was introduced for the first time in (Padberg and Rinaldi, 1987). (Lucena and Beasley, 1996) reviews those methods and their applications. They are used to solve $\mathcal{NP}$-hard problems, such as the traveling salesman problem (Padberg and Rinaldi, 1987), the cardinality constrained knapsack problem (de Farias Jr. and Nemhauser, 2003), the generalized assignment problem and the capacitated p-median problem (Vasil'ev, 2009). They are also very commonly used in commercial softwares, such as CPLEX (IBM, 2015), and open source softwares, such as GLPK (Makhorin, 2015). The generation of cuts used in branch-and-cut methods are generally derived from cutting plane algorithms. For an ILP, the Gomory's cuts (Gomory, 1958) and the Chvátal-Gomory cuts (Chvátal, 1973) are well known. For the knapsack problem, Crowder et al. (1983) introduced valid inequalities, called the cover inequalities.

The interest for the branch-and-cut methods in a multi-objective context is very recent. Jozefowiez et al. (2012) proposes an enumerative single-objective branch-and-cut method for the minimum labeling Hamiltonian cycle problem and two of its variants. The problem considers two objective functions. The second one is bounded with a small range of values and can only take integer values. On the same principle than for the ε-constraint method, this objective function is replaced by an inequality constraint. The algorithm firstly considers an initial value for the right hand side of the inequality, creating a subproblem and solve this subproblem by a dedicated branch-and-cut method. The value on the second objective function of the weakly efficient solutions found during the solving of the subproblem are used to determine the next right hand side to use in the inequality derived from the objective function.

Multi-objective branch-and-cut methods have been presented in recent conferences.

A very recent work on an optimized parallel branch-and-cut method for the bi-objective travel salesman problem has been presented at the Recent Advances in Multi-Objective Optimization

workshop, an article on the subject should be published soon (Stidsen and Andersen, 2015).

In (Gadegaard et al., 2015), the authors propose to add cuts to the computation of upper bound sets based on a weighted sum of the linear relaxation, in a branch-and-bound method designed for bi-objective combinatorial optimization problems.

1.4.5 Dynamic programming

Dynamic programming (Bellman, 1957) is an implicit enumeration method, often designed for a specific problem. They have been used to solve multi-objective optimization problems, such as the shortest path problem (Martins, 1984), knapsack problems ((Klamroth and Wiecek, 2000), (Captivo et al., 2003), (Bazgan et al., 2009a) and (Figueira et al., 2013)), among others.

For example to solve a knapsack problem, dynamic programming methods represent the feasible solutions using an oriented acyclic graph, in which there exists a one to one association between the paths and the feasible solutions. For each node, the corresponding nondominated points are sought thanks to a recursive formula, using the nondominated points of its predecessors in the graph. On a similar principle than for the branch-and-bound method, bound sets can be used to reduce the number of nodes to consider.

1.4.6 Two phase method

A two phase method is a general principle introduced by (Ulungu and Teghem, 1995) to solve bi-objective MOCO problems. As its name indicates, this method searches the efficient solutions in two phases: the extreme supported efficient solutions firstly and then all the others. We present only the bi-objective version of the two phase method, however the method has been extended for three objectives and more (for MOCO problems (Przybylski et al., 2010a), for MOMILP problems (Vincent, 2013)). The extension to more than two objectives is not straightforward since the natural order is lost.

The first phase aims to find X_{SE1_m}, the dichotomic method presented in 1.4.1 of Aneja and Nair (1979) is generally employed.

The second phase consists in finding the non-supported and the non-extreme supported efficient solutions. In the bi-objective context, the natural order of the nondominated points allows to characterize the localization of such solutions. Indeed, the non-supported and supported non-extreme nondominated points are located in the triangles defined by two adjacent supported nondominated points (according to the natural order). Let y^r and y^l be two adjacent supported nondominated points according to the natural order, with $y^l_1 < y^r_1$ (and thus $y^l_2 > y^r_2$ due to the natural order), we define the triangle $\triangle(y^r, y^l)$ composed of the points y^r, y^l and (y^l_1, y^r_2). Those triangles are illustrated in Figure 1.10. The point (y^l_1, y^r_2) called a *local nadir point*.

The second phase consists in investigating all triangles to find the non-supported nondominated points and the non-extreme supported nondominated points, using a multi-objective solution method. During this phase, the search space is reduced to the triangle investigated and a solution method specific to the problem to solve is executed. It can be a branch-and-bound method ((Visée et al., 1998) for the bi-objective knapsack problem or (Vincent, 2013) for MOILP problems), a dynamic programming method ((Delort and Spanjaard, 2010) for the bi-objective knapsack problem) or a ranking method ((Przybylski et al., 2008) for the bi-objective assignment problem, (Jorge, 2010) for the bi-objective knapsack problem). The ranking method is a single-objective solution method which searches the k best solutions.

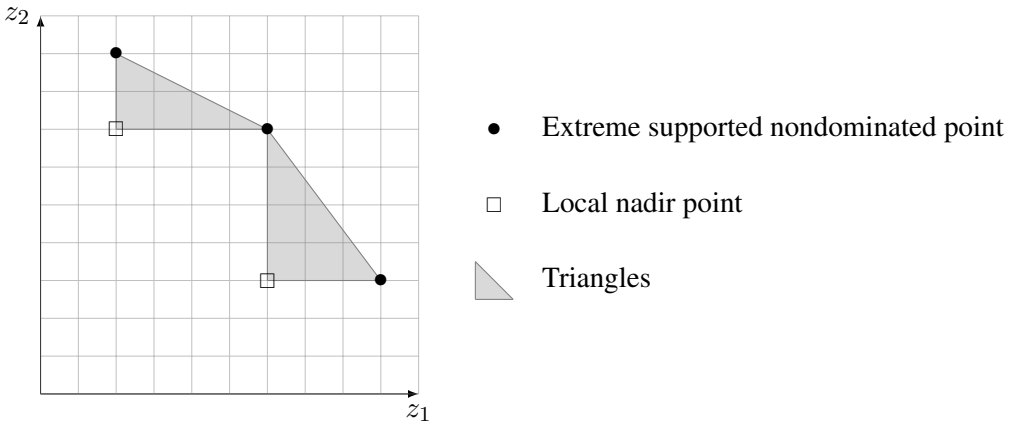


Figure 1.10: Illustration of the triangles $\triangle(y^r, y^l)$

1.4.7 Approximation methods

When dealing with difficult problems, the exact solution methods can be highly time consuming, in particular when the instance has a large number of variables. In practical situations, the size of the instances to solve can be larger than what an exact solution method can handle. The aim of an approximation method is to provide a good approximation of the optimal solution (or efficient solutions) within a reasonable time. One could think about stopping a branch-and-bound or dynamic programming method after a given execution time, however this technique does not provide any guarantee on the quality of the obtained solutions. An approximation method offers a tradeoff between the quality of the obtained solutions and the time and memory complexity of the algorithm. In this section, we present several approximation solution methods, divided in two categories: approximation schemes; heuristic and metaheuristic methods.

Approximation schemes

An approximation scheme guarantees the quality of the approximation returned by the algorithm, prior to the execution. Definition 15, from Erlebach et al. (2001), formalizes this guarantee in the single and multi-objective context.

Definition 15 ((Erlebach et al., 2001))**.**

- *For $\rho \geq 0$, a solution x^1 is called a* ρ-approximation of a solution x^2 *if* $z_k(x^1) \geq \dfrac{z_k(x^2)}{\rho}$ *for all* $k = 1, \ldots, p$.
- *A set $\bar{X}$ of feasible solutions for a problem P is called a* ρ-approximation of X_E *if, for every solution $x \in X_E$, $\bar{X}$ contains a feasible solution $\bar{x}$ that is a ρ-approximation of x.*
- *An algorithm that runs in polynomial time in the size of the input and that always outputs a ρ-approximation of X_E is called a* ρ-approximation algorithm.
- *A* polynomial-time approximation scheme *(PTAS) for X_E is a family of algorithms that contains, for every fixed constant $\epsilon > 0$, a $(1+\epsilon)$-approximation algorithm A_ϵ.*
- *If the running-time of A_ϵ is polynomial in the size of the input and in ϵ^{-1}, the family of algorithms is called a* fully polynomial-time approximation scheme *(FPTAS).*

For a multi-objective optimization problem, a ρ-approximation method guarantees a maximal gap between the approximation solutions and the efficient solutions, which is independent from the range of the coefficients.

In (Safer and Orlin, 1995a), the authors highlight necessary and sufficient conditions for the existence of FPTAS for MOCO problems. In (Safer and Orlin, 1995b), they prove the existence of a FPTAS for several network flow, knapsack and lot-sizing problems. An improved FPTAS for the multi-objective shortest path problem is proposed in (Tsaggouris and Zaroliagis, 2006).

Heuristic and metaheuristic methods

In (Reeves, 1993), the authors defined heuristic methods as methods seeking a good approximation of the efficient solutions, without any guarantee, at a low computational cost. It leads quickly to solutions of good quality, however it rarely finds an efficient solution (or optimal solution depending on the context) and is often stuck into local optima. A heuristic is often problem specific, however they can be classified based on the approach used.

The greedy heuristics construct a solution from scratch by iteratively setting variables, bringing immediately the best value according to the objective function. For example, when dealing with a traveling salesman problem, the route can be iteratively built by systematically going to the nearest neighbor.

The local search methods are based on a different approach. Knowing a "good" feasible solution (obtained by a greedy heuristic for example), they investigate its neighborhood hoping to find a better solution. A neighborhood is a set of solutions differing only by a few characteristics. For example, the neighbors of a solution can be the solutions differing on only one variable. The local search method have received a particular attention and there exist several variants of those methods. Among others, we can cite the Pareto local search (see (Hoos and Stützle, 2004) or (Dubois-Lacoste, 2014) for reviews), the iterated Pareto local search (see (Lourenço et al., 2003) for a review), the two phase pareto local search ((Paquete and Stützle, 2003) and (Lust and Teghem, 2010) for the bi-objective traveling salesman problem) and the indicator-based local search (Basseur and Burke, 2007).

In (Glover and Kochenberger, 2003), the metaheuristics are defined as "solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space". Contrary to the heuristics, the metaheuristics are generally applicable to several problems. Metaheuristic methods are widely used for multi-objective problems. In (Ehrgott and Gandibleux, 2000), the metaheuristics are classified into two main categories: the methods of local search in objective space (simulated annealing, tabu search, greedy randomized adaptive search, variable neighborhood search, path relinking, etc) and the population based methods (generic programming, ant colony systems, evolutionary methods, etc). Many articles deal with metaheuristic methods in the multi-objective context, among them the evolutionary algorithm have received a particular attention (see (Deb, 2015) for a review). The most well-know general evolutionary method may be NSGA-II (Deb et al., 2002).

The tradeoff between the exploitation and exploration (also called intensification and diversification) is an important concept for both heuristic and metaheuristic methods. Indeed without exploitation (intensification), the quality of the solutions remains low and without exploration (diversification) the method could be stuck into local optimum. If in general the local search methods offers a fast convergence to efficient solutions, they need to be guided to cover the set of nondominated points. On contrary the evolutionary algorithms do not need any guide, but can converge slowly to efficient solutions. In order to take advantage of the two approaches, hybrid metaheuristic methods have been proposed, a review is presented in (Ehrgott and Gandibleux, 2008). The

matheuristic methods mix (Maniezzo et al., 2010) metaheuristic methods and exact solution methods. For example, the algorithm introduced in (Gandibleux and Fréville, 2000) is a tabu search in which cuts are introduced to solve the multi-objective knapsack problem.

1.5 Conclusion

Due to their theoretical and practical complexity, the multi-objective combinatorial optimization (MOCO) problems have received a lot of attention in the literature. Many methods have been developed to solve them either exactly or approximately. In this thesis we focus on exact solution methods for MOCO methods. Most of the methodologies developed in this context are general. However, in order to achieve a good practical efficiency, their algorithmic instantiation is specific and exploit the problem structure. It is in particular the case for branch-and-bound problems. The majority of the MOCO problems for which there exists an exact solution method are problems whose single-objective version can be solved in polynomial or pseudo polynomial time. Then the solution methods rely on the convex relaxation of the problem, as it is for example the case in two phase methods and branch-and-bound methods. However, there exists only a few solution methods, efficient in practice, for problems whose single-objective version is more difficult to solve. Different components should also adapt to this increased difficulty. In this thesis, we intend to analyze the branch-and-bound method components in this case and to design a solution method efficient in practice.

Among the components to adapt, the computation of the upper bound set is the first to analyze, since its impact on the performances is important. The convex relaxation may be too expensive to be used if the single-objective version of the problem is difficult in practice. Then other relaxations should be used. The relaxations could be specific to the considered problem or generic (as the linear programming relaxation). However, using a generic relaxation might lead to large upper bound sets and it may be desirable to introduce cuts to prune earlier subproblems. When introduced in the context of a branch-and-bound method, the resulting methodology is a branch-and-cut method, for which the interest in multi-objective contexts is very recent.

In the literature, the attention is mainly centered around the upper bound sets and the other components of branch-and-bound method are less considered. For example, the separation strategies and the choice of the active node are generally static. However, designing dynamic strategy could allow a better adaptation to the problem and the different instances and thus improve the solution method.

In this thesis, three main components are analyzed: the upper bound set, the separation strategy and the generation and exploitation of cuts.

In order to be provided with reference works and in absence of similar studies, the different algorithms are presented and tested on the case study of knapsack problems.

2

Knapsack problems

Knapsack problems are reference combinatorial optimization problems, for two reasons: (1) a large range of real life applications can be formalized as knapsack problems and (2) knapsack problems appear as subproblems in the mathematical formulation of many more complex problems. As a consequence, the knapsack problem is one of the most studied combinatorial optimization problems in the operations research community. Over the years, many contributions have been brought on knapsack problems, such as fundamental results and a large number of dedicated solution methods. Moreover, knapsack problems are also widely used as benchmark for generic algorithms, such as heuristic or metaheuristic methods.

The multi-objective version of the problem also raises a particular interest. Two variants emerge, uni-dimensional and multi-dimensional knapsack problems. These variants are used to elaborate dedicated solution methods and also to evaluate generic algorithms.

This chapter presents different variants of knapsack problems and key components of the dedicated solution methods, which will be necessary for the following of the manuscript.

2.1 Definition

The variant of the knapsack problem that we will investigate in this thesis is the most general one: the multi-dimensional multi-objective knapsack problem ($pOmDKP$), defined below. As its name indicates, it considers multiple objective functions and multiple constraints (also called dimensions). However, in this chapter we will also be interested in simpler variants, considering fewer objective functions and/or constraints. Indeed, the first solution methods for knapsack problems were historically dedicated to those simpler variants.

The interest accorded to the different variants of knapsack problems comes from their practical applications, such as capital budgeting or allocating processors (see Martello and Toth (1990), Kellerer et al. (2004) and da Silva et al. (2004)). In (Clausen et al., 2010), the two-dimensional knapsack problem is used to solve the problem of assigning seats in a train for a group of people traveling together. In (Aisopos et al., 2013), the knapsack problem is used to model resource management in software.

The $pOmDKP$ consists in choosing items such that the capacity on each dimension is not exceeded, while maximizing the objective functions. Each item $j \in \{1, \ldots, n\}$ has a weight $w_{ij} \in \mathbb{N}$ on each dimension $i \in \{1, \ldots, m\}$ and a profit $c_j^k \in \mathbb{N}$ on each objective $k \in \{1, \ldots, p\}$. $\omega_i \in \mathbb{N}^*$ is the capacity on the dimension $i \in \{1, \ldots, m\}$.

The mathematical formulation of the $pOmDKP$ is thus:

$$\begin{array}{lll} \max & \sum_{j=1}^{n} c_j^k \, x_j & k = 1, \ldots, p \\ s.t. & \sum_{j=1}^{n} w_{ij} \, x_j \leq \omega_i & i = 1, \ldots, m \\ & x_j \in \{0, 1\} & j = 1, \ldots, n \end{array} \qquad (pOmDKP)$$

As in (Kellerer et al., 2004), to avoid trivial situations and without lost of generality, we assume that:

- $w_{ij} \leq \omega_i \ \forall j \in \{1, \ldots, n\} \ \forall i \in \{1, \ldots, m\}$, i.e. no item exceeds the capacity of any dimension, otherwise the item j could never be selected.
- $\sum_{j=1}^{n} w_{ij} > \omega_i, \ \forall i \in \{1, \ldots, m\}$, i.e. all the items cannot be selected all together on one dimension without exceeding the capacity, otherwise the constraint i would be redundant.
- $\sum_{i=1}^{m} w_{ij} > 0, \ \forall j \in \{1, \ldots, n\}$, i.e. it is allowed for an item to have a null weight on a dimension, as long as it does not have a null weight on all dimensions. Otherwise the item j would trivially always be selected.
- $\sum_{k=1}^{p} c_j^k > 0, \ \forall j \in \{1, \ldots, n\}$, similarly to the previous point, it is allowed for an item to have a null profit on an objective function, as long as it does not have a null profit on all objective functions. Otherwise the item j would trivially never be selected.

The knapsack problem knows many variants, depending on the number of objectives and dimensions considered. We denote those variants $pOmDKP$ with p being the number of objective functions to maximize and m the number of dimensions. If $p = 1$ then "$1O$" is omitted in the denomination of the problem, similarly if $m = 1$ "$1D$" is omitted. The classical single-objective uni-dimensional knapsack problem is thus denoted KP. KP being $\mathcal{NP}$-hard, all the variants of the $pOmDKP$ are also $\mathcal{NP}$-hard (Kellerer et al., 2004). The next section details the characteristic of each variant and the solution methods designed to solve them. The reader can also refer to different reviews on knapsack problems: (Kellerer et al., 2004, Fréville, 2004, Jalali Varnamkhasti, 2012) and (Lust and Teghem, 2012), in which applications of knapsack problems and other variants (which are not in the scope of the thesis) are presented.

2.2 Single-objective uni-dimensional knapsack problem

The KP is the most classic version of $pOmDKP$ and the easiest since it only considers one constraint and one objective function. To simplify the definitions given in this section, the formu-

lation of the problem is stated as follows:

$$\begin{array}{lll} \max & \displaystyle\sum_{j=1}^{n} c_j\, x_j & \\ \text{s.t.} & \displaystyle\sum_{j=1}^{n} w_j\, x_j \leq \omega & \\ & x_j \in \{0,1\} & j = 1,\ldots,n \end{array} \qquad (KP)$$

2.2.1 Utility and bound

This problem has the interesting property that its LP relaxation is particularly easy to solve. A simple algorithm to solve this LP relaxation is given in (Dantzig, 1957). The items are ordered according to the non-increasing order of the profit-to-weight ratio, also called the utility of an item (Definition 16).

Definition 16 (Utility of an item)**.** *Let us consider the item* $j \in \{1,\ldots,n\}$*. The* utility *of this item is* $u_j = \dfrac{c_j}{w_j}$.

When the items are ordered in non-increasing order of the utilities, the optimal solution x^{LP} of the LP relaxation is obtained as follows. The first items are selected, until the residual capacity is not enough to select the next item completely. This latter item is called the *break item*. A fraction of the break item is added to the solution, such that it fills the residual capacity. Thus we obtain the following solution:

$$\begin{cases} x_j^{LP} = 1 & j = 1,\ldots,b-1 \\ x_b^{LP} = \dfrac{\omega - \sum_{j=1}^{b-1} w_j}{w_b} & \\ x_j^{LP} = 0 & j = b+1,\ldots,n \end{cases}$$

where b is the index of the break item, $b = \min\left\{ k : \displaystyle\sum_{j=1}^{k} w_j > \omega \right\}$.

Since the coefficients and the variables of the knapsack problem are integer, then $\lfloor z^{LP} \rfloor$ is also an upper bound for the optimal solution.

The solution $\hat{x}$ such that:

$$\begin{cases} \hat{x}_j = 1 & j = 1,\ldots,b-1 \\ \hat{x}_j = 0 & j = b,\ldots,n \end{cases}$$

is a feasible solution for the KP problem. Thus it can be used as a lower bound for the objective value of the optimal solution. We denote $\hat{c} = \displaystyle\sum_{j=1}^{b-1} c_j$ its profit and $\hat{w} = \displaystyle\sum_{j=1}^{b-1} w_j$ its weight.

In (Martello and Toth, 1977), the authors proposed an upper bound for the optimal value, based on $\hat{x}$. Instead of adding a portion of the break item, they analyze two situations: the break item is entirely selected ($x_b = 1$) or it is not selected at all ($x_b = 0$). The upper bound is:

$$u^{MT} = \max\left\{ \left\lfloor \hat{c} + (\omega - \hat{w})\, \frac{c_{b+1}}{w_{b+1}}) \right\rfloor, \left\lfloor \hat{c} + c_b + (\omega - \hat{w} - w_b)\, \frac{c_{b-1}}{w_{b-1}} \right\rfloor \right\}$$

The corresponding solution is either:

$$\begin{cases} x_j^{MT} = 1 & j = 1, \ldots, b-1 \\ x_b^{MT} = 0 & \\ x_{b+1}^{MT} = \dfrac{\omega - \hat{w}}{w_{b+1}} & \\ x_j^{MT} = 0 & j = b+2, \ldots, n \end{cases} \text{ or } \begin{cases} x_j^{MT} = 1 & j = 1, \ldots, b-2 \\ x_{b-1}^{MT} = 1 - \dfrac{\hat{w} + w_b - \omega}{w_{b-1}} & \\ x_b^{MT} = 1 & \\ x_j^{MT} = 0 & j = b+1, \ldots, n \end{cases}$$

The authors proved that $u^{MT} \leq \lfloor z^{LP} \rfloor$.
Example 3 computes for an instance of KP the optimal solution of the LP relaxation and u^{MT}.

Example 3 (LP relaxation of a KP)**.** *We consider the following instance of KP, where the item are ordered in the non-increasing order of the utilities.*

$$\begin{array}{lrl} \max & 20\,x_1 + 7\,x_2 + 8\,x_3 + 10\,x_4 + 7\,x_5 & \\ \textit{s.t.} & 2\,x_1 + x_2 + 7\,x_3 + 13\,x_4 + 11\,x_5 \leq 17 & (KP\text{-}1) \\ & x_j \in \{0,1\},\ j = 1, \ldots, 5 & \end{array}$$

$\min\left\{k : \sum_{j=1}^{k} w_j > 17\right\} = 4$, *thus* x_4 *is the break item. The value of the optimal solution of the LP relaxation is:* $z^{LP} = 20 + 7 + 8 + \dfrac{17 - (2+1+7)}{13}\,10 = 35 + \dfrac{7}{13}\,10 \approx 40.384$ *and the corresponding solution is* $x^{LP} = (1, 1, 1, \dfrac{7}{13}, 0)$.

$\lfloor z^{LP} \rfloor = 40$ *is also an upper bound for KP-1.*

$\hat{x} = (1, 1, 1, 0, 0)$, $\hat{c} = 35$ *and* $\hat{w} = 10$. $\hat{x}$ *is a lower bound for the optimal value.*

The upper bound from (Martello and Toth, 1977) is $u^{MT} = \max\{\lfloor 35 + (17-10)\,\dfrac{7}{11}) \rfloor, \lfloor 35 + 10 + (17 - 10 - 13)\,\dfrac{8}{7} \rfloor\} = \max\{39, 38\} = 39$. *The solution corresponding to this upper bound is* $x^{MT} = (1, 1, \dfrac{1}{7}, 1, 0)$.

The optimal solution of this instance is $(1, 1, 0, 1, 0)$ *of objective value 37.*

2.2.2 Core concept

The ordering of the items according to the utilities is used not only to compute the optimal solution of the LP relaxation, but also in many solution methods dedicated to the KP. Indeed the utility of an item reflects the interest to select this item and more particularly its tendency to be selected in an optimal solution. An item with very high utility would be almost certainly selected in an optimal solution, while an item with a very low utility would be almost certainly not selected.

In (Balas and Zemel, 1980), the authors observed that, for random instances, the optimal solution of a KP instance varies from the optimal solution of its LP relaxation by only a few variables, whose indexes are generally near to the break item (if the items are ordered according to the utilities). They introduced the notion of core. Based on the knowledge of the optimal solution x^*, the core is a set of items $x_j, j = n_1, \ldots, n_2$, with $n_1 = \min\{j, x_j^* = 0\}$ and $n_2 = \max\{j, x_j^* = 1\}$. We have $x_j^* = 1$ for $j = 1, \ldots, n_1 - 1$; $x_j^* = 0$ for $j = n_2 + 1, \ldots, n$; and the KP problem can be reduced to the core problem KP_C:

$$
\begin{aligned}
\max \quad & \sum_{j=n_1}^{n_2} c_j\, x_j + \sum_{j=1}^{n_1-1} c_j \\
\text{s.t.} \quad & \sum_{j=n_1}^{n_2} w_j\, x_j \leq \omega - \sum_{j=1}^{n_1-1} w_j \\
& x_j \in \{0,1\} \qquad\qquad j = n_1, \ldots, n_2
\end{aligned}
\tag{KP_C}
$$

Obviously the optimal solution is not known in advance. However, several exact solution methods for the KP are based on this notion of core. The optimal solution is found by successively approximating the core. The core problem being smaller than the original KP problem, its optimal solution is easier to find.

2.2.3 Solution methods

Various methods have been proposed to solve exactly KP. One of the most efficient in practice is *Combo* presented in (Martello et al., 1999). It is a dynamic programming method using the notion of core. It is an improvement of *Minknap* (Pisinger, 1994). The time complexity of *Combo* is pseudo polynomial in $O(n\omega)$.

More recently, parallel algorithms have been investigated, for example in (Boyer et al., 2012) the method introduced is a parallel dynamic programming method.

A different approach of the KP has been presented in (Hifi and Mhalla, 2013), presenting a sensitivity analysis of the optimal solution regarding the variation of weights of the items.

All KP instances have not the same practical difficulty, even for the same number of variables. The ratio $\frac{\omega}{\sum_{j=1}^{n} w_j}$, called the tightness ratio, is often used as a difficulty indicator. It is commonly accepted that the KP instances with a tightness ratio around 0.5 are the most difficult to solve.

2.2.4 Preprocessing treatments

It is well known that the computational time required to solve the KP increases with the number of variables. Therefore, preprocessing treatments are often applied, prior to the solution method, to reduce the number of variables. One of the first variable reduction procedure dedicated to KP has been presented in (Ingargiola and Korsh, 1973). The method consists in creating subproblems by fixing one variable x_j to 0 and 1 for each $x_j,\ j = 1, \ldots, n$. An upper bound is computed for each of those subproblems and compared to the best feasible solution found (corresponding to the lower bound). If the objective value of this upper bound on the subproblem is lower than the objective value of the lower bound then the variable can be fixed to its opposite value. The upper bound based on the LP relaxation is particularly suitable for this method, since it is easy to compute and differs only slightly among the subproblems. We can remark that even if this variable reduction procedure has been defined specifically for the KP problem, it can easily be adapted to any other problem as long as an upper bound is available. This preprocessing treatment has been adapted to knapsack problems with more objective functions and/or more constraints.

2.3 Single-objective multi-dimensional knapsack problem

In practical problems several, resources can be limited. It is to represent this situation that the knapsack problem considering several dimensions and one objective function ($mDKP$) is studied. It is also called the d-dimensional knapsack (d-KP) in (Kellerer et al., 2004).

The $mDKP$ is considerably more difficult than KP (Fréville, 2004). The $mDKP$ with $m \geq 2$ is proven to have no FPTAS algorithm in (Gens and Levner, 1979), however it has a PTAS algorithm.

Many solutions methods have been designed specifically to solve this problem, both in an exact and approximate approach. Even if this manuscript focuses on exact approaches, we can note that many heuristic and metaheuristic methods have been developed for the $mDKP$. For example, in (Hanafi and Fréville, 1998) a tabu search based on the surrogate relaxation is presented; in (Boyer et al., 2009) two heuristic methods are presented, both based on the surrogate relaxation of $mDKP$; Khemakhem et al. (2012) presents a hybrid metaheuristic mixing a tabu search method and neighborhood search; Glover (2013) presents a greedy algorithm using the surrogate relaxation. A review on the heuristic and metaheuristic methods to solve this problem can be found in (Jalali Varnamkhasti, 2012).

2.3.1 Solution methods

Solving exactly the $mDKP$ is the subject of many works. Most of the solution methods for this problem are branch-and-bound methods. The first branch-and-bound method designed to solve the $mDKP$ was in (Thesen, 1975). The LP relaxation is less often used in a solution method for this problem than for the KP problem. Indeed, it is more complex to solve and the optimal solution of the LP relaxation often have more than one variable with a fractional value.

Few years later Shih (1979) provides another branch-and-bound method for the $mDKP$. The upper bound is based on the problems considering only the constraint i, for $i = 1, \ldots, m$. The LP relaxation of those m problems is solved and the minimum value obtained defines the upper bound of the problem.

In the branch-and-bound algorithm introduced in (Gavish and Pirkul, 1985)), the upper bound is based on the surrogate relaxation of the problem. Experiments have been conducted on different separation strategies (including some using the surrogate multipliers) and on strategies to update the surrogate multipliers. Procedures to reduce the number of constraints and the numbers of variables are performed as a preprocessing treatment and during the algorithm.

Several dynamic programming methods have also been designed to solve the $mDKP$. In (Weingartner and Ness, 1967) two approaches of dynamic programming are presented. The first one starts with an empty knapsack and iteratively adds items in it, while the second one starts with an infeasible knapsack in which all items are selected and it iteratively removes items from it.

The concept of core has been generalized for the $mDKP$ by Puchinger et al. (2006). Different generalizations of the utility of an item are tested to order the items for the core problem.

The special case of the $2DKP$ has received a particular attention. In (Fréville and Plateau, 1996), a branch-and-bound method is designed for which the bounds are based on the surrogate dual of the problem (Fréville and Plateau, 1993). Another branch-and-bound algorithm is presented in (Martello and Toth, 2003) using jointly the Lagrangian, surrogate and LP relaxations. Schulze et al. (2013) consider the $2DKP$ by transforming one of the constraints into an objective function. The problem becomes a bi-objective optimization problem with one constraint. Since the coefficients of the new objective function are negative, it is a special variant of $2OKP$. They implemented a dynamic programming method to solve it. In (Schulze and Klamroth, 2015) this method is generalized to the $3DKP$.

2.3.2 Preprocessing treatments

As for the KP, preprocessing treatments are applied to reduce the number of variables. Since the practical complexity of the $mDKP$ also increases with the number of constraints, preprocessing treatments also aim to reduce the number of constraints.

In (Gavish and Pirkul, 1985), the authors adapt the preprocessing treatment introduced in (Ingargiola and Korsh, 1973) to the multi-dimensional case. The upper bound used is based on the LP relaxation.

Another preprocessing treatment is presented in (Fréville and Plateau, 1994) to reduce the number of variables and constraints of the problem, based on the dual surrogate problem. The procedure presented in (Osorio et al., 2002) to fix variables is also based on the surrogate relaxation.

In the preprocessing treatment presented in (Balev et al., 2008), another relaxation is examined: the LP relaxation. The procedure aims to fix subsets of variables by comparing the upper and lower bounds of subproblems. The upper bound is based on the LP relaxation and the lower bound is computed by a dynamic programming principle.

In (Hill et al., 2012), the Lagrangian relaxation and core concepts are used to reduce the number of variables.

2.3.3 Surrogate relaxation

We can remark that the surrogate relaxation of the $mDKP$ is both used in exact and approximation solution methods, as well as in some preprocessing treatments to reduce the number of variables and/or constraints. This relaxation is one of the most used when dealing with $mDKP$. Indeed, by aggregating the constraints, the resulting problem is a KP problem, for which there exist exact pseudo-polynomial time algorithms (for example (Martello et al., 1999)).

As already explained in Section 1.3.3, the dual surrogate problem gives the tightest upper bound based on the surrogate relaxation for $mDKP$. However, solving the surrogate dual is $\mathcal{NP}$-hard (Boyer, 2007). Several methods have been developed to solve the dual surrogate for $mDKP$ (for example (Karwan and Rardin, 1979) and (Dyer, 1980)). (Boros, 1986) has shown that the number solution of knapsack problems required to been solved is linear in the number of considered constraints. (Fréville and Plateau, 1993) presents an exact solution method for the surrogate dual problem in the bi-dimensional case.

The particular structure of $2DKP$ (two constraints) allows to define the surrogate problem with only one multiplier by normalizing the multiplier $\mu \in \mathbb{R}^2_{\geq}$. In (Fréville and Plateau, 1993) the authors use the normalization presented in (Gavish and Pirkul, 1985). In this manuscript, we will use another normalization to explain the algorithm proposed by (Fréville and Plateau, 1993).

The surrogate problem for $2DKP$ can be formulated using a multiplier $u \in [0,1]$:

$$\begin{array}{ll} \max & c_j\, x_j \\ \text{s.t.} & u \sum_{j=1}^{n} w_{1j}\, x_j + (1-u) \sum_{j=1}^{n} w_{2j}\, x_j \leq u\, \omega_1 + (1-u)\, \omega_2 \\ & x_j \in \{0,1\},\ j = 1, \ldots, n \end{array} \qquad (SR(u))$$

The problems $SR(\mu)$ with $\mu \in \mathbb{R}^2_{\geq}$ and $SR(u)$ with $u \in [0,1]$ are equivalent with $u = \dfrac{\mu_1}{\mu_1 + \mu_2}$. We have chosen to use the same notation SR for the definition using a vector μ and the one using a scalar u for the sake of simplicity. In the remainder of the manuscript,

only the definition using a scalar is used. The constraint of the problem $SR(u)$ is called the *surrogate constraint* and u the *surrogate multiplier*. We denote x^u the optimal solution of the surrogate relaxation $SR(u)$.

In (Fréville and Plateau, 1993), the authors remark that x^u, with $u \in [0,1]$ cannot violate both constraints simultaneously. Indeed, if both constraints are violated, then the surrogate constraint is violated too. Moreover, if x^u does not violate any of the constraints, then it is feasible for the $mDKP$ problem and thus it is the optimal solution of the $mDKP$ problem. Thus when x^u the optimal solution of $SR(u)$ is not feasible for the $mDKP$ problem, then it violates one and only one of the two constraints of the $mDKP$ problem.

The interval of surrogate multipliers for which the solution x^u is feasible for the surrogate relaxation can be calculated and it depends on which constraint is violated by x^u, it is given in Proposition 2.

Proposition 2. *We define*

$$v(x^u) = \frac{\omega_2 - \sum_{j=1}^{n} w_{2j}\, x_j^u}{\sum_{j=1}^{n} w_{1j}\, x_j^u - \sum_{j=1}^{n} w_{2j}\, x_j^u - \omega_1 + \omega_2}$$

- *If x^u violates the first constraint of the $mDKP$ problem then x^u is feasible for every surrogate relaxation $SR(u')$ with $u' \in [0, v(x^u)]$.*
- *If x^u violates the second constraint, then x^u is feasible for every surrogate relaxation $SR(u')$ with $u' \in [v(x^u), 1]$.*

The algorithm *SADE* (Fréville and Plateau, 1993) is based on this observation to compute SD the surrogate dual. They perform a dichotomic method on the values of the surrogate multipliers. At any step of the algorithm, if the optimal solution x^u of a surrogate problem $SR(u)$ (with $u \in [0,1]$) is feasible for the $mDKP$ problem, then $z(SD) = z(SR(u))$. The dual surrogate is found so the algorithm stops. For the remaining of the explanation, to avoid redundancy, we do not repeat this condition, it is applied implicitly.

During the algorithm an interval $[u^l, u^r]$ is refined such that there exists $u \in [u^l, u^r]$ with $z(SR(u)) = z(SD)$. If the optimal solution $x^{1/2}$ of $SR(1/2)$ violates the first constraint of the $mDKP$ problem, the interval $[u^l, u^r]$ is initialized to $[v(x^{1/2}), v(x^1)]$, x^1 being the optimal solution of $SR(1)$. If on the contrary $x^{1/2}$ violates the second constraint of the $mDKP$ problem, the interval $[u^l, u^r]$ is initialized to $[v(x^0), v(x^{1/2})]$, x^0 being the optimal solution of $SR(0)$.

If $u^l > u^r$ the algorithm stops and we have $z(SD) = \min(z(SR(u^l)), z(SR(u^r)))$. If $u^l \leq u^r$ then $SR\left(\frac{u^l + u^r}{2}\right)$ is solved. If its optimal solution x^* satisfies the first constraint, then the interval is refined to $[v(x^*), u^r]$, otherwise it is refined to $[u^l, v(x^*)]$.

The algorithm *SADE* is proven to be correct and to converge in a finite number of steps (Fréville and Plateau, 1993). The authors observed that in practice the number of steps executed is independent of the number of variables in the $mDKP$ problem.

2.4 Multi-objective uni-dimensional knapsack problem

Since the middle of the 90's, the $pOKP$ has gained interest. The $pOKP$ is considerably more difficult than the KP. Indeed, we no longer search for one optimal solution but for several efficient

solutions. Moreover, the notion of utility of items, which led to a key component of the solution methods for the KP, is more difficult to define for the $pOKP$. Indeed, for each objective function a utility can be computed. Since the objective functions are often conflicting, the utility of a same item can vary drastically regarding the considered objective.

The $pOKP$ is #P-complete and intractable. In (Safer and Orlin, 1995b), the authors show that there exists a FPTAS for the $pOKP$. Bazgan et al. (2009b) also developed a FPTAS for this problem.

2.4.1 Solution methods

Even if $pOKP$ has received a lot of attention for the elaboration or the test of metaheuristic methods ((Gandibleux and Fréville, 2000) for a tabu search method, (da Silva et al., 2007) for a scatter search method), many exact solution methods have been developed for this problem. Generally, the solution methods employed are generalizations of single-objective solution methods.

Dynamic programming methods have been developed for the special case of $2OKP$ (for example Captivo et al. (2003), Delort and Spanjaard (2010), Figueira et al. (2013)), as well as for the general $pOKP$ (for example Klamroth and Wiecek (2000) and Figueira et al. (2010)). In (Figueira et al., 2015), the authors aim to reduce the memory consumption of a dynamic programming method for the $2OKP$.

The core concept, which has proven its practical efficiency for the KP, is generalized to $pOKP$ in (da Silva et al., 2008).

Several branch-and-bound methods have also been developed to solve the $pOKP$. This section describes especially those methods which are in the scope of this thesis, along with two phase methods.

The branch-and-bound method presented in (Ulungu and Teghem, 1997) is one of the first algorithms specifically designed for $2OKP$. In this method, the upper bound is the point whose value on the objective function $k = 1, \dots, p$ is the upper bound of (Martello and Toth, 1977) u^{MT} for the single-objective version of the problem, considering only the objective function k.

In (Visée et al., 1998) a branch-and-bound method is embedded in a two phase method to solve $2OKP$. The single-objective version of $2OKP$ is a KP for which there exists a pseudo-polynomial time algorithm to solve it (for example in (Martello et al., 1999)). Thus the supported nondominated points are "easy" to obtain (using a dichotomic method on the weighted sum problem). Each triangle $\triangle(y^r, y^l)$ is investigated individually, with y^r and y^l are adjacent supported nondominated points. Three upper bounds are computed v^1, v^2 and v^λ, for three weighted sum problems, whose directions are respectively $(1, 0)$, $(0, 1)$ and $\lambda = (y_2^l - y_2^r, y_1^r - y_1^l)$. Each bound is u^{MT} presented in (Martello and Toth, 1977). If $v^1 < y_1^l$ or $v^2 < y_2^r$, then no feasible solution can belong to the triangle $\triangle(y^r, y^l)$. Therefore, the node is fathomed by infeasibility. The node can be fathomed by dominance under two circumstances: there exists $l \in L$ such that $l \geq (v^1, v^2)$ or for all $l \in L$, $\lambda^T l \geq v^\lambda$; L being the lower bound set of the problem.

Jorge (2010) and Delort (2011) also developed a two phase method. However, the second phase is not a branch-and-bound method but a ranking method (Jorge, 2010) and a dynamic programming method (Delort, 2011). These two Ph.D. thesis also propose preprocessing treatments to reduce the number of variables, which will be detailed in Section 2.4.2.

Jorge (2010) also presents a branch-and-bound method to solve the $3OKP$ problem. The author proposed two versions of the branch-and-bound algorithm.

In the first version, similarly to what is done in (Ulungu and Teghem, 1997), three upper bounds (y^1, y^2 and y^3) are computed, for the problems considering only, respectively, the first, the

second and the third objective function. Each of these bounds is u^{MT} presented in (Martello and Toth, 1977). Then the point $y^U = (y_1^1, y_2^2, y_3^3)$ is defined. If this point does not allow to prune the subproblem, then the weighted sum problem with direction $\lambda = y^U$ is solved exactly; we denote y^λ the obtained solution. The upper bound set is defined by $\{y \in \mathbb{R}^p : y_1 \leqq y_1^U, y_2 \leqq y_2^U, y_3 \leqq y_3^U, \lambda^T y \leqq \lambda^T y^\lambda\}_N$ is used for the dominance test. The search-tree is built according to a depth first search strategy. The variables are fixed according to a static order based either on an aggregation of the utilities (average, maximum or minimum of the utility, detailed in Section 3.1.1), on a dominance relation on the utilities or on the dominance rank of the items. The author remarks that the branching strategy giving the best results is not always the same, even for instances built with the same generator.

In the second version of (Jorge, 2010), the upper bound set is the convex relaxation of the problem. The choice of the active node is a dynamic strategy. Five criteria are considered lexicographically to determine the active node; three on them consider the structure of the subproblem (tightness ratio, capacity and number of variables) while the two others concern the upper bound set (number of supported solutions and number of supported solutions belonging to the lower bound set). The branching strategy is also dynamic. If there exists a variable which has the same value in all the solutions of the lower bound set, then this variable is chosen for the branching. Otherwise the static strategy of the first version is applied.

2.4.2 Preprocessing treatments

Preprocessing treatments have been elaborated to reduce the number of variables of the instances to solve. Indeed, some variables have the same value in all the efficient solutions. An item selected in all efficient solutions is called a *mandatory* item and an item not selected in any efficient solution is a *forbidden* item (using the terminology of (Delort, 2011)). The aim of those treatments is to identify and fix those variables in order to obtain a smaller instance and thus speed up the solving. Ideally we would like to identify all mandatory and forbidden items, using preprocessing treatments. However, this task is difficult. In a two phase method, the preprocessing treatments can be achieved at two levels: before the first phase (global preprocessing treatment) or during the second phase before the solving in each triangle (local preprocessing treatment). When executed during the second phase, the preprocessing treatment generally allows to fix more variables. Indeed, there are only a few mandatory and forbidden variable when considering the whole objective space. However, they are more when considering a restricted area, like a triangle when the preprocessing is done during the second phase.

In the following of this section, we omit the indexes relative to the dimension since only one dimension is considered in a $pOKP$. Thus the problem is defined as follows:

$$\begin{array}{lll} \max & \sum_{j=1}^{n} c_j^k \, x_j & k = 1, \ldots, p \\ s.t. & \sum_{j=1}^{n} w_j \, x_j \leq \omega & \\ & x_j \in \{0,1\} & j = 1, \ldots, n \end{array} \qquad (pOKP)$$

In (Jorge, 2010), the preprocessing treatment is a global preprocessing treatment, based on two concepts: the bounds on the cardinality of an efficient solution and the dominance relations between items. The first notion was introduced in (Glover, 1965) for the KP and extended to the $pOKP$ in (Gandibleux and Fréville, 2000); it is presented in Definition 17.

Definition 17 (Cardinality of an efficient solution (Gandibleux and Fréville, 2000))**.** *Let* $e(x) = \sum_{j=1}^{n} x_j$ *be the cardinality of an efficient solution* $x \in X_E$*. Then* $LB(\omega) \leq e(x) \leq UB(\omega)$ *with:*

- $LB(\omega) = \max s : \sum_{j=1}^{s} w_j \leq \omega$, (w_j) *sorted in decreasing order.*
- $UB(\omega) = \max s : \sum_{j=1}^{s} w_j \leq \omega$, (w_j) *sorted in increasing order.*

In (Jorge, 2010), the items are compared to each other using a relation of dominance. To do so a vector $v^j = (c_j^1, \ldots, c_j^m, -w_j)$ is associated to each item $j = 1, \ldots, n$. An item $j \in \{1, \ldots, n\}$ dominates an item $l \in \{1, \ldots, n\}$ if $v^j \geq v^l$, i.e. if $c_j^k \geq c_l^k, k = 1, \ldots, m$ and $w_j \leq w_l$. Informally an item dominates another one if the profit associated to this item is higher than for the other one, while the weight is lower. For each item, the set of its dominated items is defined as well as the set of items dominating it (see Definition 18).

Definition 18 (Preferred and dominated sets (Jorge, 2010))**.** *Let* $j \in \{1, \ldots, n\}$.

- *The preferred set of* j *is* $Pref(v^j) = \{l \in \{1, \ldots, n\} : v^l \geq v^j\}$.
- *The dominated set of* j *is* $Dom(v^j) = \{l \in \{1, \ldots, n\} : v^j \geq v^l\}$.

Based on this definition and on the cardinality of an efficient solution, Jorge (2010) highlights four propositions, that are used by the preprocessing treatment. Those propositions are given in Proposition 3.

Proposition 3 ((Jorge, 2010))**.** *Let* $j \in \{1, \ldots, n\}$.

- *If* $|Pref(v^j)| \geq UB(\omega)$*, then* $x_j = 0, \ \forall x \in X_{E_M}$.
- *If* $\sum_{l \in Pref(v^j)} w_l + w_j > \omega$ *then* $x_j = 0, \ \forall x \in X_{E_M}$.
- *If* $n - |Dom(v^j)| \leq LB(\omega)$*, then* $x_j = 1, \ \forall x \in X_{E_M}$.
- *If* $\sum_{l \notin Dom(v^j)} w_l + w_j \leq \omega$ *then* $x_j = 1, \ \forall x \in X_{E_M}$.

The preprocessing treatment used in (Visée et al., 1998) and (Delort and Spanjaard, 2010) is a local preprocessing treatment, considering successively each variable and each value for it. A subproblem is created differing from the global problem by only one fixed variable j. The variable is fixed to a value $v \in \{0, 1\}$. The upper bound set U for $\bar{Y_N}$ (subproblem) is computed and compared to the lower bound set L for Y_N (global problem). The comparison of bound sets is similar to the branch-and-bound method. If $(U - \mathbb{R}^p_{\geqq}) \cap (l + \mathbb{R}^p_{\geqq}) = \emptyset$ (i.e. U is "below" L) then there cannot exist any efficient solution with $x_j = v$. Thus for all efficient solution, we have $x_j = 1 - v$. Thus this variable can be fixed to the value $x_j = 1 - v$, without altering the set of efficient solutions.

This preprocessing treatment fulfills two goals: the reduction of the set of variables and initialize the lower bound set with feasible solutions

2.5 Multi-objective multi-dimensional knapsack problem

By considering simultaneously several constraints and several objective functions, $pOmDKP$ gathers the difficulties of $pOKP$ and $mDKP$. This induces a real difficulty to elaborate practical

efficient exact solution methods. In (Erlebach et al., 2001), the authors present an PTAS for the $pOmDKP$. A review on this problem can be found in (Lust and Teghem, 2012).

Due to its difficulty, many works have developed metaheuristics for $pOmDKP$. In (Zitzler and Thiele, 1999), the authors considered the special case of $pOmDKP$ where the number of objective functions equals the number of dimensions and conceived an evolutionary algorithm. Jaszkiewicz (2004) and Ishibuchi et al. (2009) present evolutionary algorithms for $pOmDKP$. In (da Silva et al., 2004), the authors designed an evolutionary algorithm based on the surrogate relaxation for $2OmDKP$. A local search is implemented in (Tricoire, 2012).

The next section focuses on exact methods developed to solve the $pOmDKP$.

2.5.1 Solution methods

The types of methods employed to solve the $pOmDKP$ are mostly the same as for $mDKP$, i.e. core concept methods, branch-and-bound method and dynamic programming methods.

Mavrotas et al. (2009) and Mavrotas et al. (2011) extended the concept of core to $2OmDKP$ and proposed an exact solution method based on this concept. A more generic core-based method is implemented for $pOmDKP$ in (Lust and Teghem, 2012).

A branch-and-bound method is developed in (Florios et al., 2010) to solve the special case of $3O3DKP$. Its upper bound set for $\bar{Y_N}$ of a subproblem is the ideal point of the linear relaxation. To initialize the lower bound set, the LP relaxation of the global problem is solved; then the fractional variables in the extreme efficient solutions are rounded creating integer solutions; among those solutions, the feasible ones are added to the lower bound set. In the experiments of (Florios et al., 2010), four branching strategies are tested and compared to a random branching strategy. In the first one, the variables are ordered based on the extreme efficient solutions of the LP relaxation of the global problem. The values of the variables on those solutions are summed and the branching fixes uppermost the variable with the highest sum. The three other branching strategies are based on the utilities defined for each pair objective-constraint, considering either the average value of all utilities, either its maximum, either a lexicographic order of the utilities. According to the experiments, the authors note that the branching strategies are essential for the practical efficiency of the algorithm and that the first branching strategy seems to give the best results.

The authors remarked that the branch-and-bound method is too time consuming to solve bigger instances, thus they proposed two evolutionary algorithms.

The special case $2O2DKP$ was the focus of the work of Gandibleux and Perederieieva (2011). A dynamic programming method is developed, using upper bound sets based on the LP relaxation and the surrogate relaxation (detailed in Section 2.5.2). The dominance relation defined in (Bazgan et al., 2009a) are adapted to $2O2DKP$ and used during the dynamic programming method. Three orders have been compared for the items. The first one is the composite branching proposed by Florios et al. (2010), in the second one the variables are ordered according to the average of the utilities (one for each pair objective-constraint) and in the third one they are ordered according to the maximum utility value and the equalities are broken thanks to the average of the utilities.

2.5.2 Surrogate relaxation

As it was the case for $mDKP$, the surrogate relaxation is also used in approximation methods (da Silva et al., 2004) and exact methods (Gandibleux and Perederieieva, 2011) for $pOmDKP$.

In (da Silva et al., 2004), the surrogate relaxation is used in the diversification and intensification components of the evolutionary algorithm to convert the $2OmDKP$ into a $2OKP$. The authors remark that the convex relaxation of the surrogate relaxation, for a given multiplier, can

be close to the supported efficient solutions of $2OmDKP$ problem for some parts of the objective space, but far away from it on other parts. Thus they aim to find a set of multipliers in order to obtain a tight upper bound set, they use a subgradient-like method to find those multipliers.

In (Gandibleux and Perederieieva, 2011), the upper bound set is based on the LP relaxation and the surrogate relaxation. The convex relaxation of the surrogate relaxation of the problem is computed using several surrogate multipliers. The upper bound sets obtained are merged, using Proposition 1 (page 32), to obtain a tighter upper bound set, this upper bound set is called a *surrogate family*. The upper bound set obtained by the LP relaxation is merged with the surrogate family bound set. The authors remark that the tightness of the upper bound set obtained increases with the number of surrogate multipliers employed, however the computational time required to its computation also increases. In order to maintain an interesting tradeoff between the computational time and the quality of the upper bound set, the authors select a restricted set of well spread surrogate multipliers. Based on the experimental results, a set of three multipliers seems be a good tradeoff for their method.

2.6 Conclusion

In most of the works presented in this chapter, the order in which the items are considered impacts the practical efficiency of the methods, both in single and multi-objective contexts, no matter the number of considered dimensions. However, the difficulty of finding a strategy leading to good performances depends on the number of objectives and dimensions. In the classic KP variant, sorting the items according to their utility generally leads to efficient methods in practice. For the other variants ($mDKP$, $pOKP$ and $pOmDKP$), several utilities are defined for each item and their values for a same item generally vary a lot, so it is more difficult to exploit this information. The elaboration of "good" strategies to determine the order in which the items are considered has arisen a lot of interest. Many different strategies have been elaborated (using or not the notion of utility of items, they are generally static) and have been compared. It has been observed that the strategy offering the best result is not necessarily the same on all instances.

Even if the classes of solution methods are the same for the different variants of the knapsack problem (branch-and-bound method, dynamic programming method, etc), the components of those methods differ regarding the considered variant. This is in particular true between single and multi-objective problems. For instance, when dealing with bi-dimensional knapsack, the surrogate relaxation plays an important role. For $2DKP$, there exists a practically efficient method (Fréville and Plateau, 1993) to compute the dual surrogate, i.e. computing the tightest upper bound based on the surrogate relaxation. For $2O2DKP$, several surrogate multipliers are used to improve the quality of the obtained upper bound sets (Gandibleux and Perederieieva, 2011). Even if the authors remark that the quality of the upper bound set increases with the number of considered multipliers, there is no guarantee to find the tightest upper bound set based on this relaxation.

On the other hand, in a bi-objective context, the convex relaxation leads to good performance in practice when the single-objective version of the problem is solvable in polynomial or pseudo-polynomial time, as it is the case for $2OKP$. However, this is not the case for $2OmDKP$, so the performance obtained by this relaxation might be less interesting.

Among the classes of employed methods, the branch-and-cut method is absent, despite its practical efficiency when solving combinatorial optimization problem in the single-objective context. The aim of this thesis is to elaborate an efficient branch-and-cut method for the knapsack problem considering simultaneously several objective functions and several constraints. In the first part on this manuscript, we investigate the branching strategies (determining the order in

which the items are considered) and try to elaborate a dynamic strategy. In the second part, we focus on the computation of a tight upper bound set for $2O2DKP$ based on the surrogate relaxation and propose a generalization of the dual surrogate in the multi-objective context. The last part deals with the elaboration of a branch-and-cut method for the same problem.

3

Branching strategies for the bi-objective knapsack problem

In the previous chapters, we have observed that many works investigated the order in which the variables are considered in a solution method, in particular for branch-and-bound and dynamic programming methods. Most of the strategies are static. The definition of an order for variables is generally linked to the structure of the problem. For knapsack problems, the utilities of an item reflect the interest of selecting it. Therefore, the order in which the variables are considered in a solution method for knapsack problems relies on this notion of utility. Obviously, the information emerging from the utilities of the items is easier to exploit for the version of the knapsack with only one objective function and one constraints, since a single utility per item can be defined. When the number of objective functions and/or constraints increases, utilities can be defined for each pair objective-constraint and the value obtained for a same item can be very different. In this chapter, we consider the problem $2OKP$. Many orders for variables have been proposed for this problem but no real study has been conducted to compare them.

In this chapter, we aim to elaborate a practically efficient branching strategy in a branch-and-bound method to solve the $2OKP$. We firstly compare different static branching strategies, which have been defined for $2OKP$ or adapted from other variants of the knapsack problem. Obviously, we consider orders in the context of branch-and-bound methods, but also in the context of dynamic programming methods. The second part of this chapter consists in the definition of a dynamic branching strategy for the $2OKP$, which is tested experimentally on a benchmark of instances.

3.1 Introduction

As already mentioned, for a $pOmDKP$, a utility can be defined for each pair objective-constraint for each item. Thus for the particular case of $pOKP$, p utilities are defined for each item $j \in \{1, \ldots, n\}$ by $u_j^k = \frac{c_j^k}{w_j}$ for $k = 1, \ldots, p$.

The next section presents different orders elaborated for the consideration of the variables in solution methods. We only describe orders based on the notion of utilities.

3.1.1 Orders of variables and branching strategies in the literature

We can remark that a branching strategy can be defined by any order of the variables. Thus, we do not present only the strategies introduced for branch-and-bound methods, but also the orders defined for dynamic programming methods, since they can be adapted to define a branching strategy. In the literature, no work has highlighted a specific order for the variables as the best for every possible instance and it is probably impossible to elaborate such a static order.

To obtain a single order based on several utilities, an aggregation of the utilities can be used. In (Ulungu and Teghem, 1997), the authors define an order $\mathcal{O}^k$ for each objective function $k = 1, 2$. In $\mathcal{O}^k$, the variables $j \in \{1, \ldots, n\}$ are sorted in decreasing order of the utilities u_j^k, for $k = 1, 2$. The *rank* r_j^k of an item $j \in \{1, \ldots, n\}$ is the position of the item j in the order $\mathcal{O}^k$, $k = 1, 2$. The ranks are then summed and the items sorted according to the increasing order of this sum.

In (Bazgan et al., 2009a), the definition of the orders $\mathcal{O}^k$ in extended to $pOKP$. The authors consider three aggregations of the ranks, aiming to find a compromise between the orders $\mathcal{O}^k$, $k = 1, \ldots, p$.

The first one is the one from (Ulungu and Teghem, 1997), it sorts the items in increasing order of the sum of the ranks (the value associated to the item j is $\sum_{k=1}^{p} r_j^k$). According to this strategy, the first variable considered is the one having the lowest average rank on $\mathcal{O}^k, k = 1, \ldots, p$.

The second branching strategy presented in (Bazgan et al., 2009a) sorts the items in increasing order of the maximum of the ranks defined for this item. In case of equality, the sum of the ranks is used to break ties. The value associated to each item $j = 1, \ldots, n$ and use to sort the items is the following:

$$\max\{r_j^k, k = 1, \ldots, p\} + \frac{\sum_{k=1}^{p} r_j^k}{p\,n}.$$

The items are thus sorted according to their worst rank.

The last branching strategy is symmetric to the previous one. We associate to each item $j = 1, \ldots, n$ its best rank, i.e.

$$\min\{r_j^k, k = 1, \ldots, p\} + \frac{\sum_{k=1}^{p} r_j^k}{p\,n}.$$

The items are then sorted in increasing order of their best rank and the equalities are broken by the average rank in $\mathcal{O}^k, k = 1, \ldots, p$.

These three branching strategies presented in (Bazgan et al., 2009a) are also compared in (Jorge, 2010), with other branching strategies. Among those strategies, k ones are defined by using each $\mathcal{O}^k, k = 1, \ldots, p$ as a branching strategy. The item first fixed in the algorithm is then the most promising item, according to the considered objective function $k \in \{1, \ldots, p\}$. We do not describe here the other orders in this chapter.

In a two phase method, the orders of the items may depend on the triangle investigated. In (Visée et al., 1998), when investigating the triangle $\triangle(y^r, y^l)$, the search direction $\lambda = (y_2^l - y_2^r, y_1^r - y_1^l)$ weights the utilities u^1 and u^2 in a weighted average. The items are considered in decreasing order of the weighted average. This order has been also used in (Delort, 2011) and compared to the order based on the maximum rank of the items defined in (Bazgan et al., 2009a) and a random order on the variables.

The orders in which the variables are considered in a solution method have also been defined for $pOmDKP$. In (Florios et al., 2010), three orders based on a scalarization of the utilities are tested as branching strategies for the $3O3DKP$. A utility is defined for each pair objective function/dimension. The first order considers only the maximum of the utilities defined and the items are sorted in decreasing order of this value. The second one deals with the average value of the utilities, again the items are sorted in decreasing order of this value. In the last one, the variables are sorted according a lexicographic order of the utilities, in decreasing order.

In (Gandibleux and Perederieieva, 2011), two branching strategies are used. The first one is the branching strategy based on the average of the utilities presented in (Florios et al., 2010). The second one is based on the maximum of the utilities of an item, similarly to the one presented in (Florios et al., 2010), except that in case of equality, the average of the utilities is used to break the ties.

The different separation strategies that we have presented are very diverse. Some of them take into account the value of the utilities of the items, some the rank of the items in the orders $\mathcal{O}^k$, $k = 1, \ldots, p$. The orders considering the value of the utilities can be sensitive to the range of the coefficients. For example, an objective function can have higher coefficients than another and would then impact the ordering more that another objective function. This is less the case for the order based on the direction defined by triangles.

Despite those differences, all the orders select the "best" variable (the most promising variable) in first position. This paradigm aims to increase the chances of obtaining efficient solutions or at least solutions of good quality early in the execution of the algorithm, thus allowing to prune nodes or states earlier in the remaining of the execution.

Each of the previous works compare a restricted number of branching strategies. Usually they compare few new branching strategies with one or two branching strategies from the literature, which were assessed to give the best performance. In all the studies, no order lead to a higher practical efficiency than the others on all instances, even if some are better on average.

3.1.2 Branching strategies of the study

In this section, we compare a set of 23 branching strategies for $2OKP$, that we will call branching heuristics. Among them, there will be branching strategies corresponding to the orders presented in Section 3.1.1, but also new ones. In particular, we do not consider only the "best variable first" paradigm but also the "worst variable first" paradigm. This second paradigm aims to determine early in the solution method the variables that will never be selected in efficient solutions. The name of a branching heuristic will be composed of the criterion used to sort the items, followed by "best" if it follows the "best variable first" paradigm, "worst" otherwise.

The first branching strategy is the random branching heuristic *Rand*, used as a reference for the comparisons. Since the instances we use (described in Section 3.3) are generated randomly, the random order is the order of the variables in the instance file.

Table 3.1 presents the 22 other heuristics used in this section. The best variable first strategies are presented in the columns (3) and (4) and the worst variable first strategies are in columns (5) and (6). The column (2) indicates the criterion according to which the variables are sorted and columns (4) and (6) the order used for the strategy on this criterion. The names of the strategies are given in column (3) and (5).

The simplest branching heuristics (Lines 1 and 2 of Table 3.1) order the items only according to their value on the objective functions, without taking into account their weight. On the same idea of considering only one of the two objective functions, four branching heuristics (Lines 3 and

		Best variable first strategies		Worst variable first strategies	
(1)	(2) Criterion for $j = 1, \ldots, n$	(3) Name	(4) Order	(5) Name	(6) Order
1	c_j^1	*Z1-best*	decreasing	*Z1-worst*	increasing
2	c_j^2	*Z2-best*	decreasing	*Z2-worst*	increasing
3	u_j^1	*U1-best*	decreasing	*U1-worst*	increasing
4	u_j^2	*U2-best*	decreasing	*U2-worst*	increasing
5	$\min(u_j^1, u_j^2)$	*Min-best*	decreasing	*Min-worst*	increasing
6	$\max(u_j^1, u_j^2)$	*Max-best*	decreasing	*Max-worst*	increasing
7	$\frac{u_j^1 + u_j^2}{2}$	*Avg-best*	decreasing	*Avg-worst*	increasing
8	$\frac{\lambda_1 u_j^1 + \lambda_2, u_j^2}{\lambda_1 + \lambda_2}$	*Triang-best*	decreasing	*Triang-worst*	increasing
9	$r_j^1 + r_j^2$	*SumRank-best*	increasing	*SumRank-worst*	decreasing
10	$\min(r_j^1, r_j^2) + \frac{r_j^1 + r_j^2}{2\,n}$	*MinRank-best*	increasing	*MinRank-worst*	decreasing
11	$\max(r_j^1, r_j^2) + \frac{r_j^1 + r_j^2}{2\,n}$	*MaxRank-best*	increasing	*MaxRank-worst*	decreasing

Table 3.1: Branching heuristics of the study. $\lambda = (y_2^l - y_2^r, y_1^r - y_1^l)$ for the investigated triangle $\triangle(y^r, y^l)$.

4) are defined using either the utility u^1 either u^2. The utilities u^1 and u^2 can also be aggregated by a minimum, maximum, average or weighted average function (Lines 5 to 8). The last separation heuristics are based on the rank of the variables on $\mathcal{O}^1$ and $\mathcal{O}^2$ and the aggregations used in (Bazgan et al., 2009a) (Lines 9 to 11).

Example 4. *Let us consider the following $2OKP$ instance:*

$$\begin{array}{lrcl} \max & 11\,x_1 + 2\,x_2 + 8\,x_3 + 10\,x_4 + 9\,x_5 + x_6 & & \\ \max & 2\,x_1 + 7\,x_2 + 8\,x_3 + 4\,x_4 + x_5 + 3\,x_6 & & \\ \textit{s.t.} & 4\,x_1 + 4\,x_2 + 6\,x_3 + 4\,x_4 + 3\,x_5 + 2\,x_6 & \leq & 11 \\ & x_j \in \{0,1\},\ j = 1, \ldots, 6 & & \end{array} \qquad (2OKP\text{-}2)$$

The utilities and rank are given in Table 3.2.

Table 3.2 gives few examples of orders defined by branching heuristics. The first column indicates the name of the considered branching heuristic and the second column the order on the variables.

Items j	1	2	3	4	5	6
u_j^1	2.75	0.5	1.33	2.5	3	0.5
u_j^2	0.5	1.75	1.33	1	0.33	1.5
r_j^1	2	5	4	3	1	6
r_j^2	5	1	3	4	6	2

Table 3.2: Utilities and ranks of the items for the instance $2OKP$-2.

Branching strategy	Order for the variables
U2-best	$x_2, x_6, x_3, x_4, x_1, x_5$
U2-worst	$x_5, x_1, x_4, x_3, x_6, x_2$
MaxRank-worst	$x_6, x_5, x_2, x_1, x_3, x_4$
Triang-best for $\triangle((10, 15), (21, 12))$, thus direction $(5, 9)$	$x_4, x_3, x_1, x_2, x_5, x_6$

3.2 Specification of the algorithm

The algorithm we used here is a two phase method. In the first phase, the extreme supported nondominated points are searched by a dichotomic method (Aneja and Nair, 1979) using the KP solver *Combo* (Martello et al., 1999). In the second phase, a branch-and-bound method is executed to investigate every triangle $\triangle(y^r, y^l)$ defined by the extreme supported nondominated points. The upper bound set for $\bar{Y}_N$ (the nondominated set of the subproblem) is the convex relaxation (computed by the method employed in the first phase). The lower bound set for Y_N (the nondominated set of the global problem) is based on the potential nondominated points found so far by the algorithm. In this work, the solution method aims to find X_{E_m} and does not aim to find X_{E_M}, we are not interested in equivalent solutions. Since the variables and the coefficients are integer, we use the shifted lower bound set as defined in Section 1.4.3.

The search-tree is explored following a depth-first search strategy, the separation strategy sets one variable to 1 first and 0 after. The branching strategy is the same during all the execution, it is chosen among the ones defined in Section 3.1.2. The triangles are explored in the lexicographic order, i.e. $\triangle(y^{r1}, y^{l1})$ is investigated before $\triangle(y^{r2}, y^{l2})$ if $y_1^{r1} > y_1^{r2}$ (then $y_1^{r1} < y_1^{r2}$ and $y^{l1} >_{lex} y_1^{l2}$ by definition of the triangles).

The preprocessing treatments of Jorge (2010) and Delort and Spanjaard (2010) presented in Section 2.4.2 can be applied. All along this study, two versions will be considered depending on whether preprocessing is applied or not. This will allow us to analyze the impact of this preprocessing on the performance of the branching heuristics.

During the separation procedure, the dominance relations from the preprocessing treatment of Jorge (2010) are exploited: when a variable is fixed to 0, all variables dominated by it are also fixed to 0; and symmetrically when a variable is fixed to 1, all variables dominating it are fixed to 1.

3.3 Benchmark

In this chapter, the performance of the branching strategies will be assessed on 53 instances. Their origins and the generators used for those instances are presented in this section. n is the

number of variables in the instance, i.e. the size of the instance. Table 3.3 presents the source and the size of the instances.

Name	Source	Number of variables
2KP50-11	Gandibleux and Fréville (2000)	50
2KP50-50		50
2KP100-50		100
2KPn-1A $n \in \{50, 100, 150, 200, 250, 300\}$	Visée et al. (1998)	n
2KPn-1B *2KPn-1C* *2KPn-1D* $n \in \{50, 100, 150, 200, 250, 300\}$	Degoutin and Gandibleux (2002)	n
4WnW1 $n \in \{50, 100, 150, 200, 250, 300\}$	Captivo et al. (2003)	n
F5050Ws *K5050Ws* $s \in \{01, \ldots, 10\}$	Captivo et al. (2003)	50

Table 3.3: Sources and number of variables of each instance for 2OKP

The tightness ratio is of 0.5 for all instances, except *2KP50-11* for which it is 0.11.

The generation of objective and constraint coefficients are detailed here.

– For *2KP50-11*, *2KP50-50* and *2KP100-50*, the objective coefficients are generated according to a uniform distribution in $\{30, \ldots, 100\}$ and the constraint coefficients in $\{20, \ldots, 500\}$.
– For *2KPn-1A*, the objective and constraint coefficients are generated according to a uniform distribution in $\{1, \ldots, 100\}$ and the tightness ratio is 0.5.
– For *2KPn-1B*, the coefficients on the first objective function and the constraint are generated according to a uniform distribution in $\{1, \ldots, 100\}$. The second objective is obtained by taking the objective coefficients of the first objective function in reverse order.
– For *2KPn-1C*, the constraint coefficients are generated according to a uniform distribution in $\{1, \ldots, 100\}$. The objective coefficients are repeated over several items, forming a plateau which size is generated according to a uniform distribution in $\{1, \ldots, \lfloor 0.1\, n \rfloor\}$. The objective coefficients are generated according to a uniform distribution in $\{1, \ldots, 100\}$.
– The instances *2KPn-1D*, are based on the instances of the previous category. The second objective function and the constraint are the same and the first objective function is obtained by taking the objective coefficients of the second one in reverse order.
– For *4WnW1*, the coefficients of the second objective function are generated according to a uniform distribution in $\{111, \ldots, 1000\}$ and the constraint coefficients in $\{1, \ldots, 1000\}$. The coefficients of the first objective function are said to be weakly correlated with the second objective function. c_j^1 is generated according to a uniform distribution in $\{c_j^2 - 100, \ldots, c_j^2 + 100\}$, $j = 1, \ldots, n$.
– For *F5050Ws*, the objective and constraint coefficients are generated according to a uniform distribution in $\{1, \ldots, 300\}$.
– For *K5050Ws*, the objective and constraint coefficients are generated according to a uniform distribution in $\{1, \ldots, 1000\}$.

Since some of the dynamic branching strategies presented in the following are time consuming

(Sections 3.5 and 3.6), we defined three groups considering instances with different size. The first group G_1 is composed of all instances with 150 variables or less, except *4W150W1* which is more difficult in practice. The second group G_2 contains all instances with 200 variables or less, at the exception of *4W150W1* and the last group G_3 contains all the instances presented in this section.

For the experiments presented in this chapter, we use virtual machine equipped with a Intel Xeon E5620 2.40GHz processor with 6 Go of RAM. All algorithms are implemented in C++.

3.4 Comparison of the branching heuristics

In this section, the performance of the branching heuristics presented in Section 3.1.2 are assessed on the benchmark G_3 of instances presented above.

When using the preprocessing treatments from (Jorge, 2010) and (Delort and Spanjaard, 2010), the method can run on bigger instances (with more variables). When running the algorithm without any preprocessing treatment, the size of the search-tree and the computational time are higher than when using those treatments. The difference grows with the size of the instance. The algorithm without the preprocessing treatments cannot handle four of the instances of the benchmark group G_3: *2KP300-1A, 2KP300-1C, 4W250W1* and *4W300W1*. They are thus excluded for this version of the algorithm.

The time required to compute the order of the variables is negligible in the algorithm and is more or less the same for all the branching heuristics. The size of the resulting search-tree is then a reliable measure to assess the branching heuristics. The computational time increases proportionally with the size of the search-tree. The evaluation criteria used to assess the performance of the branching heuristics will only rely on the size of the search-tree, without considering the computational time. Three criteria are considered.

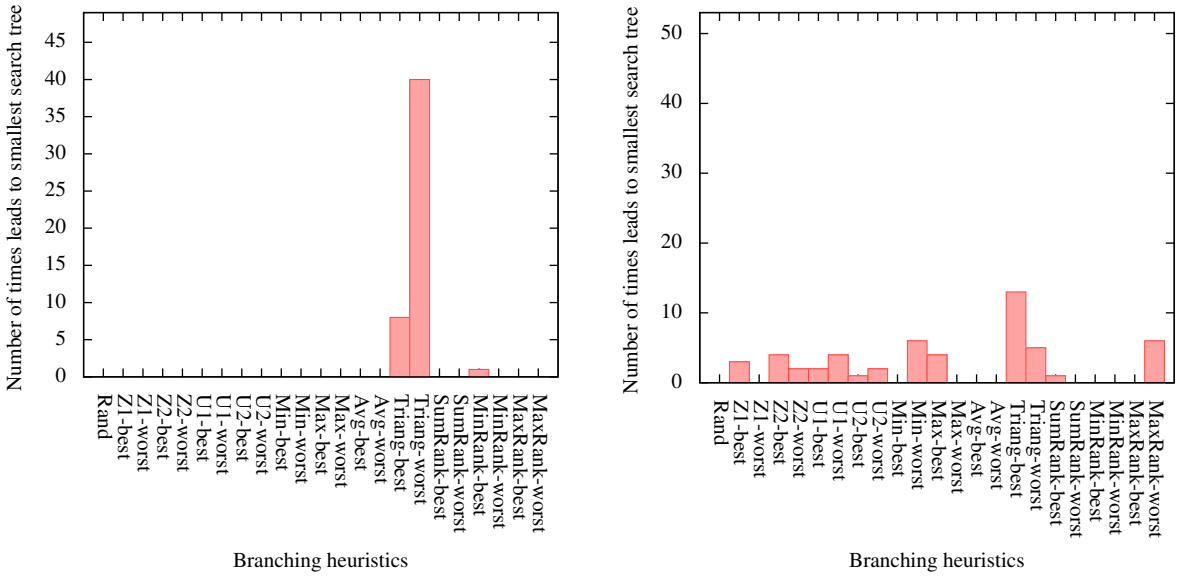
The first evaluation criterion assigns to each branching heuristic the number of instances for which it leads to the smallest search-tree (among all the branching heuristics). The results are presented in Figure 3.1. The y-scale is normalized to the number of instances considered (49 when not using preprocessing, 53 otherwise).

When no preprocessing is used (Figure 3.1a), we can see a large difference between the branching heuristics, for this evaluation criterion. The branching heuristic *Triang-worst* gives the smallest search-tree 40 times over the 49 instances. The second best heuristic according to this evaluation criterion is *Triang-best*, giving the smallest search-tree on 8 instances.

When using the preprocessing treatments (Figure 3.1b), the performances of the branching heuristics are closer to each other. *Triang-best* is the one with the highest value on the evaluation criterion, it leads to the smallest search-tree 13 times over the 53 instances. *Min-worst* and *MaxRank-worst* are the second best, giving the smallest search-tree on 6 instances.

Figure 3.1 also shows that no branching heuristic performs better than all the others on all instances. This evaluation criterion focuses for each instance on the branching heuristic that provides the smallest search-tree. Thus for each branching strategy, the highlight is only on the instances for which the performance of the heuristic is the best. However, a branching heuristic can have very good performance (the best) on some instances and very poor on others. This criterion does not give any information on the average performance of the branching strategies.

The two following evaluation criteria aim to assess the average performance of a branching heuristic. For the first one, an order of the branching heuristic is defined for every instance. The branching heuristics are sorted in increasing order of the size of the search-tree obtained. A score

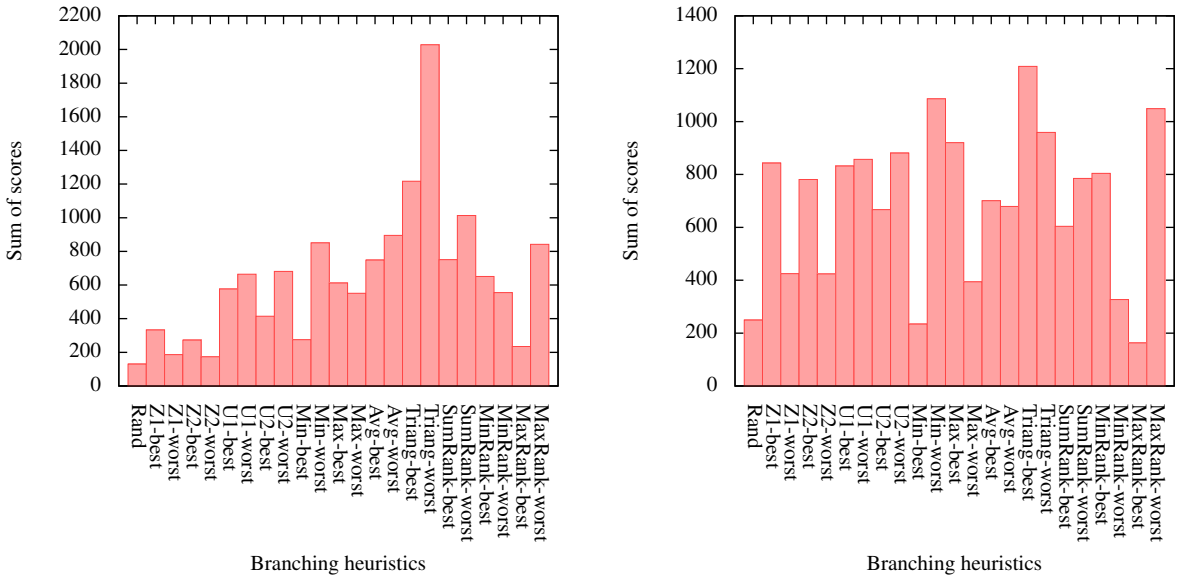


(a) Without preprocessing treatments

(b) With preprocessing treatments

Figure 3.1: Number of instances for which the heuristics give the smallest search-tree.

is associated to each branching strategy for each instance and is inversely proportional to the rank of the heuristic in the order defined according to the instance. The score equals 22 if the heuristic leads to the smallest search-tree, 21 if it leads to the second smallest, etc. Figure 3.2 shows for each branching heuristic the sum of the scores obtained on all the instances. A high sum of scores corresponds to a branching heuristic performing well in average.

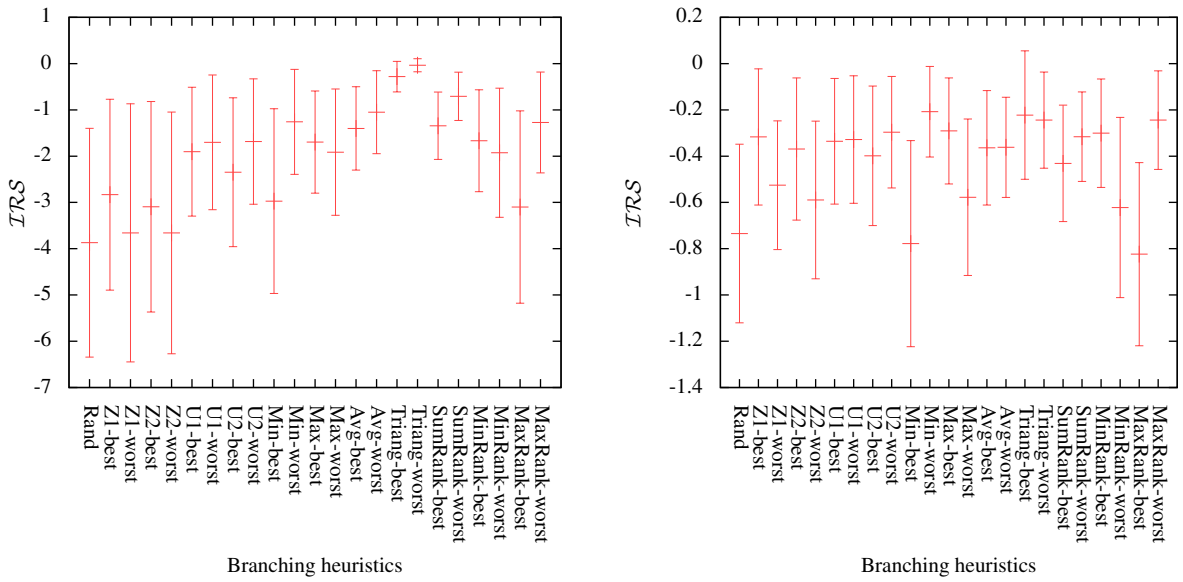


(a) Without preprocessing treatments

(b) With preprocessing treatments

Figure 3.2: Sum of the scores obtained on the instances, for each breaching strategy.

The last evaluation criterion, to assess the performances of the branching heuristics, measures for each instance the relative difference between the size of the search-tree obtained when using the branching heuristics and the size of the smallest search-tree obtained. This measure will be called the improvement ratio of the size of the search-tree ($\mathcal{IRS}$). For a given instance, we note s^t the size of the search-tree obtained by heuristic tested and s^r the size of the smallest search-tree obtained (used as a reference), we define $\mathcal{IRS} = \frac{s^r - s^t}{s^r}$. A high value of $\mathcal{IRS}$ indicates a good performance of the branching heuristic. On the contrary, a negative value indicates a degradation of the size of the search-tree. If $\mathcal{IRS}$ is equal to -2, then the size of the search-tree is twice the size of the search-tree obtained for the reference. The comparison is done with the smallest search-tree for all instances, so it is no possible to have a positive value of $\mathcal{IRS}$ here. Figure 3.3 shows for each branching heuristic the average and standard deviation of $\mathcal{IRS}$ over all instances.



(a) Without preprocessing treatments

(b) With preprocessing treatments

Figure 3.3: Average and standard deviation of $\mathcal{IRS}$ for each branching strategies.

The difference between the branching heuristics is more important when no preprocessing treatment is used, than for the version using preprocessing treatments. The branching strategies can be ordered according to the two evaluation criteria showed in Figures 3.2 and 3.3. They are given in Table 3.4.

For a given version of the algorithm, the orders defined by the two evaluation criteria are similar. However, the orders on the heuristics (Table 3.4) are very different depending on whether preprocessing is applied or not. In particular, the branching heuristic *Min-worst* has a rank 1 or 2 when the preprocessing treatments are activated but a rank 5 or 10 when they are inactive.

Usually, branching strategies use "best variable first" paradigm. However, this paradigm is not necessarily better than the opposite paradigm. For example, *Min-worst* presents better performances than *Min-best*. According to Table 3.4, the "best variable first" paradigm is better than the "worst variable first" paradigm in only half of the cases.

The branching heuristic *Rand* presents poor performance, which vary from an execution to

	Without preprocessing treatments		With preprocessing treatments	
According to	Sum of scores	$\mathcal{IRS}$	Sum of scores	$\mathcal{IRS}$
1	Triang-worst	Triang-worst	Triang-best	Min-worst
2	Triang-best	Triang-best	Min-worst	Triang-best
3	SumRank-worst	SumRank-worst	MaxRank-worst	Triang-worst
4	SumRank-best	Avg-worst	Triang-worst	MaxRank-worst
5	Avg-worst	Min-worst	Max-best	Max-best
6	Avg-best	MaxRank-worst	U2-worst	U2-worst
7	MaxRank-worst	SumRank-best	U1-worst	MinRank-best
8	MinRank-best	Avg-best	Z1-best	SumRank-worst
9	Max-best	MinRank-best	U1-best	Z1-best
10	Min-worst	U2-worst	MinRank-best	U1-worst
11	U2-worst	Max-best	SumRank-worst	U1-best
12	Max-worst	U1-worst	Z2-best	Avg-worst
13	U1-best	U1-best	Avg-best	Avg-best
14	MinRank-worst	Max-worst	Avg-worst	Z2-best
15	U1-worst	MinRank-worst	U2-best	U2-best
16	U2-best	U2-best	SumRank-best	SumRank-best
17	Min-best	Z1-best	Z1-worst	Z1-worst
18	Z1-best	Min-best	Z2-worst	Max-worst
19	MaxRank-best	Z2-best	Max-worst	Z2-worst
20	Z2-best	MaxRank-best	MinRank-worst	MinRank-worst
21	Rand	Z1-worst	Rand	Rand
22	Z2-worst	Z2-worst	Min-best	Min-best
23	Z1-worst	Rand	MaxRank-best	MaxRank-best

Table 3.4: Branching heuristics ordered according to the evaluation criteria

another. For this reason, in the rest of the manuscript, we do not include *Rand* in the branching heuristics employed.

3.5 Combination of branching heuristics by the oracle method

In the previous section, we have highlighted that, even if some branching heuristics lead to better performances in average than others, no branching heuristic reveals to be the best on all instances. The variation of performance of a branching heuristic might depend on the characteristics of the instance. In a branch-and-bound method, the characteristics of the subproblems vary from one node to another. However, the branching strategies we tested use the same branching heuristic for all separations during the solving. In this section, we wonder if using different branching heuristics during the same execution (during different separation procedures) can improve the solving, i.e. lead to a smaller search-tree and a smaller computational time. The alternating of the branching heuristics will thus be called a combination of heuristics.

An ideal method would always choose the combination of heuristics leading to the smallest search-tree, among all possible combinations. However, the search-tree being composed of a few hundreds of nodes and considering 22 heuristics, a method testing all possible combinations is not conceivable.

Instead of testing all possible combinations, we elaborated a method, called the *oracle* method,

aiming to find a good combination of heuristics, without enumerating exhaustively all the combinations. At each separation procedure, the 22 different branching heuristics are compared, the "best" one is chosen and used in the separation procedure. The "best" branching heuristic is the one leading to the best child nodes. The measure used to evaluate the quality of the nodes obtained by a branching heuristic is described in Section 3.5.1. The oracle method is a method with a visibility on the branching heuristics one step forward.

3.5.1 Evaluation of the quality of a branching heuristic

The quality of a separation can be measured on several characteristics of the two child nodes, based on the variation of the upper bound set or lower bound set for example.

Since the coefficients of the objective functions and the variables are non-negative in a $2OKP$, the feasible solutions are in $\mathbb{R}^2_{\geqq}$. The variation of the upper and lower bound sets will be measured based on the area of the zones defined by the bound sets. We denote by $\mathcal{A}(V)$ the area of the polyhedron $V \cap \mathbb{R}^2_{\geqq}$, with $V \subset \mathbb{R}^2$.

Three quality criteria

An upper bound set U specifies that the feasible solutions of the corresponding subproblem are located in $(U - \mathbb{R}^2_{\geqq}) \cap \mathbb{R}^2_{\geqq}$, which will be called the *feasibility zone*. We can remark that the feasibility zones of the child nodes are included in the one of the parent node (since the problems associated to the child nodes are subproblems of the one associated to the parent node). A separation procedure leading to a largely smaller feasibility zone in the child nodes is more likely to have a small number of descendants than if the feasibility zone is only slightly smaller.

Definition 19. *Let us consider U^0 the upper bound set of the parent node and U^1 and U^2 the ones of the child nodes. The* relative reduction of the feasible zone of the child node $l \in \{1, 2\}$ compared to the parent node *is given by the following formula.*

$$\frac{\mathcal{A}(U^0 - \mathbb{R}^2_{\geqq}) - \mathcal{A}(U^l - \mathbb{R}^2_{\geqq})}{\mathcal{A}(U^0 - \mathbb{R}^2_{\geqq})}$$

The first criterion defining the quality of a separation is the average of the relative reductions (Definition 19) of the two child nodes, it is denoted by *UB*. A separation will be considered of good quality if this value is high.

The lower bound set is common to the whole search-tree, but is updated whenever the upper bound set of a node is computed. The area $\mathcal{A}(L - \mathbb{R}^2_{\geqq})$ under the lower bound set L might increase after the computation of the upper bound sets of the child nodes, if new potentially efficient solutions are found (i.e. solution which are not dominated by solutions of the lower bound set). The probability of fathoming a node by dominance increases with the quality of the lower bound set. Thus a large difference of $\mathcal{A}(L - \mathbb{R}^2_{\geqq})$ before and after the computation of the upper bound sets of the child nodes indicates a good choice of the branching variable. This measure will be the second criterion determining the quality of a separation, it is denoted by *LB*.

The last criterion allowing us to evaluate the quality of a separation, denoted by *Other*, considers the number of potentially efficient solutions found in an other triangle than the one currently investigated. Indeed finding such solutions in an other triangle increases the quality of the lower bound set at the beginning of the solving in this triangle. Thus it increases the probability to obtain a small search-tree for this triangle. Even if those solutions do not speed up the solving in the triangle currently investigated, having a high number of potentially efficient solutions in the other triangles can be a criterion to evaluate the quality of a separation.

Quality measures

In the oracle method, at each separation, the quality measure will be used in order to determine which branching heuristic is the best. Numerous quality measures can be elaborated based on the three criteria presented above, depending on the importance given to each criterion. The quality measure is accurate if the oracle method considering this measure leads to small search trees. The performances of the different measures will be assessed with the $\mathcal{IRS}$ measure presented in Section 3.4. s^t is the size of the search-tree obtained when using the tested measure. The reference s^r is the same than in Figure 3.3, i.e. the smallest search-tree obtained for each instance. In the following, we call this reference *Best-heur*, since it corresponds to the method applying the best branching heuristic (the one giving the smallest search-tree) for each instance. This reference method is not realistic since it supposes to know a priori which one of the 22 branching heuristics would lead the smallest search-tree.

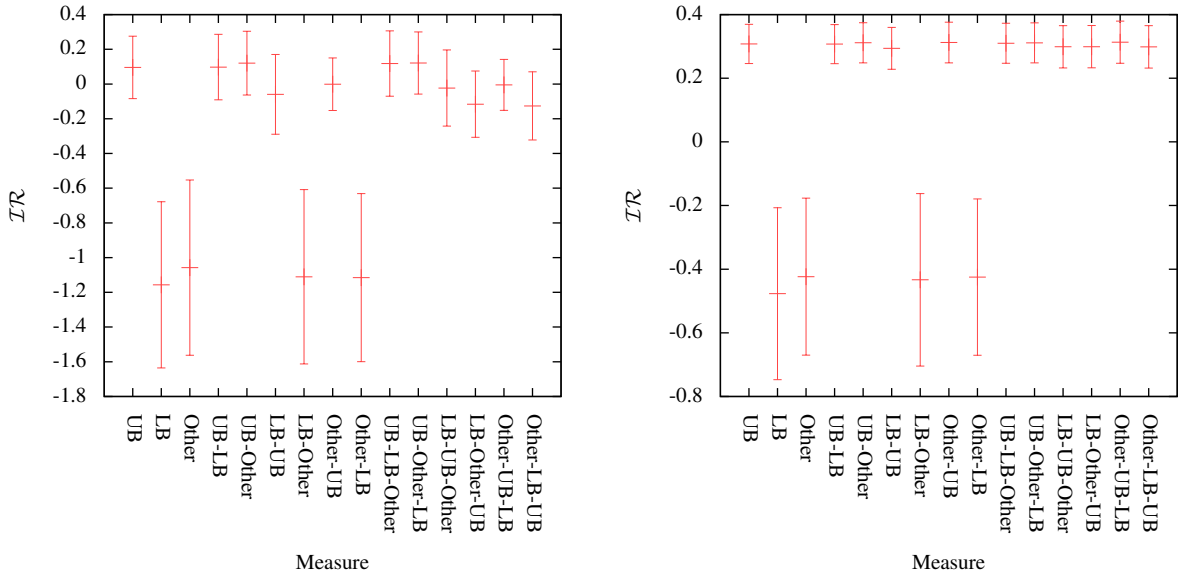
Several quality measures based on the three criteria presented have been evaluated. Due to the difference of scale and measurement unit between the quality criteria, a weighted sum on the criteria is not easy to establish and might be dependent on the range of the coefficients in the considered instance. Thus we choose to consider the criteria in a lexicographic order: the branching heuristics are ordered according to a first criterion and the equalities are broken using a second one, if the equalities persist a third criterion is used and if necessary ties are broken randomly. The components of the name of a measure are the criteria considered, in the order of importance. It can contain less than three components if some of the three quality criteria are not considered by the measure. For example, the measure *UB-Other* orders the branching heuristics according to the criterion *UB*, the equalities on this criteria are broken thanks to the criterion *Other*. For this measure, the criterion *LB* is not considered.

Figure 3.4 shows the average and standard deviation of $\mathcal{IRS}$ for the oracle method using the different measures. Figure 3.4a presents those results for the version of the algorithm in which no preprocessing treatment is used. Figure 3.4b deals with the version of the algorithm using the preprocessing treatments. Since the oracle method considers 22 branching heuristics at each separation, it is time consuming. Thus the quality measures could not be evaluated on the biggest instances presented in Section 3.3. For the version using the preprocessing treatments, the benchmark used is G_1. The benchmark is even more restricted when no preprocessing treatment is used: the 28 instances of 50 variables or less, *2KP100-1B* and *2KP100-1D* composed this benchmark.

The behavior of the quality measures is similar whether the preprocessing treatments are used or not. However, the difference is larger when no preprocessing is used than when they are used. Figure 3.4 highlights that the criterion *UB* is essential in the measure of the quality of a separation. Indeed, when it is not used, performances drop. The best performances are obtained when *UB* is the first criterion in the lexicographic order. No significative difference exists between the average $\mathcal{IRS}$ of those measures when we use preprocessing treatments, they differ from less than 0.02. However, when no preprocessing treatment is applied, the measure presenting the best performance in average is *UB-Other-LB*. Thus the most important criterion is *UB*, the equalities on this criterion are broken thanks to the criterion *Other* and then *LB*. If several branching heuristics have the same value over the three criterion, the best one is determined randomly. This measure is the one used in the oracle method, in the following.

3.5.2 The oracle method

In the previous section, we investigated the impact of the quality measure on the performances of the oracle method, regarding only the size of the search-tree. In this section, the performances



(a) Without preprocessing treatments

(b) With preprocessing treatments

Figure 3.4: Average and standard deviation of $\mathcal{IRS}$ obtained for the oracle methods using the different quality measures.

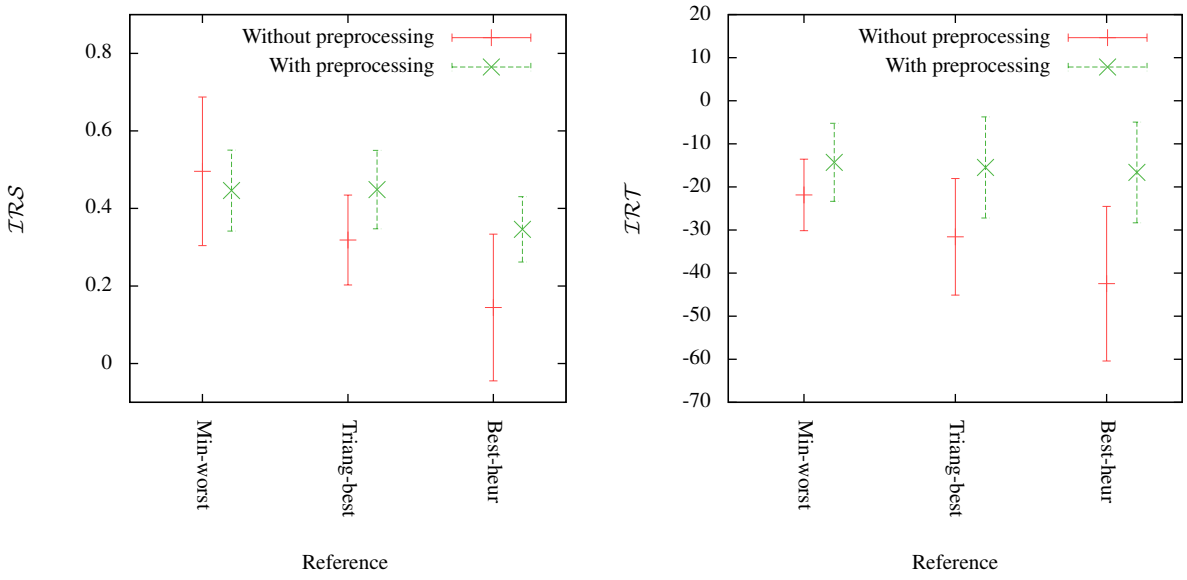
of the oracle method are analyzed with more details: comparison to three reference methods and study of the impact of the size of the instances.

Figure 3.5 shows the performances obtained for the oracle method using the *UB-Other-LB* quality measures, regarding the size of the search-tree in Figure 3.5a and the computational time in Figure 3.5b. The benchmark used is G_3 when the preprocessing treatments are applied and G_2 otherwise.

The performances of Figure 3.5a are measured by $\mathcal{IRS}$, with three different references. The two first references are algorithms applying the same branching heuristic all along the execution. We consider the two branching heuristics presenting the best performances in Table 3.4, i.e. *Min-worst* and *Triang-best*. The last reference is the one used in the previous section: *Best-heur*.

When comparing the computational time (in Figure 3.5a), the used measure is based on the same idea as $\mathcal{IRS}$. Let t^t be the computational time spent by the method to evaluate on a given instance and t^r the computational time spent by the reference method on the same instance. $\mathcal{IRT} = \frac{t^r - t^t}{t^r}$ is called the improvement ratio on the computational time. Similarly to $\mathcal{IRS}$, a positive value of $\mathcal{IRT}$ indicates that the evaluated method leads to a smaller computational time than the reference method, thus the evaluated method is better regarding the computational time. A high value of $\mathcal{IRT}$ indicates good performances of the evaluated method. Figure 3.5b presents the average and standard deviation of this measure over all instances of the benchmark, for the same three reference methods than in Figure 3.5a.

Figure 3.5a shows that the size of the search-tree is reduced on average by 34% compared to *Best-heur* and by 44% compared the branching heuristics *Min-worst* and *Triang-best*, when preprocessing treatments are used. Thus there exist combinations of branching heuristics leading to smaller search-tree than when the same branching heuristic is applied all along the branch-and-



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.5: Comparison of the oracle method and the use of one heuristic.

bound method.

However, Figure 3.5b shows that the computational time for the oracle method is 14 to 16 times larger than for the references, when the preprocessing treatments are applied. The difference of computational time is even larger when no preprocessing treatment is used. This phenomenon was expected since at each separation the oracle method tries 22 branching heuristics, compute the upper and lower bound sets, before choosing only the best heuristic according to the quality measure. Thus each separation is considerably more expensive than applying only one branching heuristic.

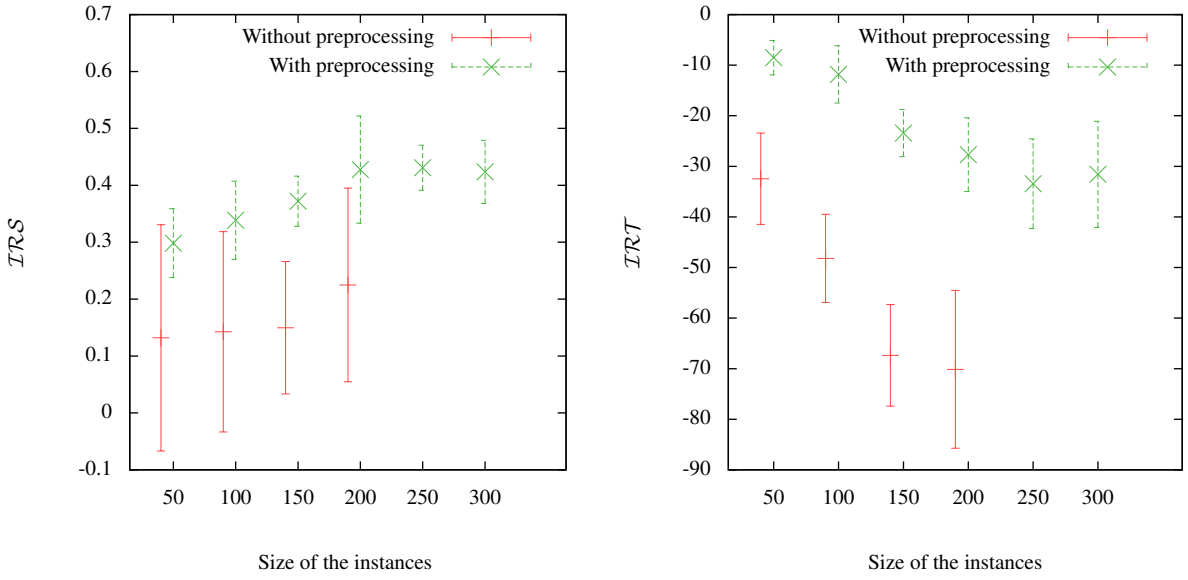
Figure 3.6 aims to highlight the impact of the size of the instances on the oracle method. The reference method taken is *Best-heur*.

In Figure 3.6, we can observe that the behavior of the oracle method varies with the size of the instances and this variation is similar whether the preprocessing treatments are used or not.

Figure 3.6a shows that the average $\mathcal{IRS}$ increases when the size of the instance increases and is between 50 and 200 variables. $\mathcal{IRS}$ seems to stagnate when the size of the instances is between 200 and 300, for the version of the algorithm using the preprocessing treatments. When the preprocessing treatments are used, the average $\mathcal{IRS}$ is between 30% and 43% for all size of instances.

The impact of the size of the instances is the opposite on the computational time (Figure 3.6b). $\mathcal{IRT}$ seems to decreases when the size of the instances increases and is between 50 and 200, then it stagnates. For all sizes of instances, the computational time is largely more important for the oracle method than for *Best-heur*.

Even if the oracle method allows to reduce significantly the size of the search-tree compared to the use of one single branching heuristic, the computational time largely exceeds the one obtained by the use of one branching heuristics. Because of the computational cost, the oracle method cannot be employed in practice to solve a $2OKP$.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.6: Comparison of the oracle method and *Best-heur*, with respect to the size of the instances.

The oracle method has shown that using a combination of branching heuristics can lead to smaller search-trees than when using the same branching heuristic all along the solving.

In this section, we combined all the branching heuristics. At each separation during the oracle method, the best branching heuristic, according to the quality measure, is chosen as the branching strategy. However, it is possible that several branching heuristics are the best according to the quality measure for a given separation, either because they choose the same branching variable, or because even if the branching variables differ the quality measure has the same value. Choosing one or the other of the branching heuristics does not make a difference for the oracle method. Since the branching heuristics are all based on the utilities of the items, this situation is not rare in practice. The 22 branching heuristics might not be necessary in the oracle method.

In the next section, we aim to highlight a subset of branching heuristics to use during the oracle method, decreasing the computational time, without compromising the quality of the separations.

3.5.3 Selection of a set of branching heuristics

We aim to improve the computational time of the oracle method by considering a subset of branching heuristics. In order to reduce the impact on the size of the search-tree obtained, this subset will be composed of the branching heuristics that give the most often the best value regarding the quality measure. To determine this subset, we apply an incremental method, on each instance. We call H the set of the branching heuristics (the 22 used in the oracle method presented above), H' the subset of branching heuristics searched. S is the set of all separations done during the oracle method, on the given instance. The incremental method assigns $\mathcal{B}_{h,s}$ to 1 if h has the best quality measure during the separation s, 0 otherwise, for $h \in H$ and $s \in S$. We recall that it is possible to have $\mathcal{B}_{h,s} = 1$ for several $h \in H$ for a given separation $s \in S$, since several branching heuristics can lead to the same branching variable. Then the method incrementally builds the set H' by executing the following steps:

1. Let $h' = \arg\max_{h \in H} \left\{ \sum_{s \in S} \mathcal{B}_{h,s} \right\}$ then $H' \leftarrow H' \cup \{h'\}$.
2. $S \leftarrow S \setminus \{s \in S, \mathcal{B}_{h,s} = 1\}$ and $H \leftarrow H \setminus \{h'\}$.
3. If $S \neq \emptyset$, go to Step 1.

We aim to find a subset of branching heuristics H' such that there is a large part of the separations for which the best quality measure is obtained by one of the branching heuristics in H'. Thus in Step 2, we delete from S the set of separations for which h' gives the best quality measure. The following of the execution focuses only on the separation for which none of the branching heuristics giving the best quality measure is in H'.

By definition of the incremental method, the first branching heuristic added in H' is more important for the oracle method, on the given instance, than the last one. In order to reflect this relative importance of the branching heuristics, a score is affected to each branching heuristic for each instance. The score is 22 for the first branching heuristic added in H', 21 for the second, etc. If a branching heuristic is not added in H', its score is 0.

During Step 1, several branching heuristics can have the same $\sum_{s \in S} \mathcal{B}_{h,s}$. Thus h' may be chosen among several branching heuristics. In order to be fair, when this situation occurs one set is created for every possible choice of h' and the method continues independently on each one of those sets. At the end of the method, the score affected to each branching heuristic is the average of the score obtained for each set built.

When this method is executed on each instance of a benchmark, the branching heuristics can be sorted according to the decreasing average score. The branching heuristics are sorted for both versions of the oracle method presented in Section 3.5.2. The orders obtained are presented in Table 3.5.

The orders presented in Table 3.5 are very similar. The branching heuristics with a good rank in those orders are more important in the oracle method (more often selected by the separation procedure) than the ones with a poor rank. Thus if we want to elaborate an oracle method using only c branching heuristics, with $c \in \{1, \ldots, 22\}$, then the c first branching heuristics in the order defined in Table 3.5 are the most suitable ones. This variant of the oracle method is called the *reduced oracle method*. It considers the first $c \in \{1, \ldots, 22\}$ branching heuristics in the order obtained for the version using the preprocessing treatments (since it is the version giving the highest value of $\mathcal{IRS}$). c is a parameter of the method. In case of equalities regarding the quality measure, the reduced oracle method favors the branching heuristics with a better rank in the order presented in Table 3.5.

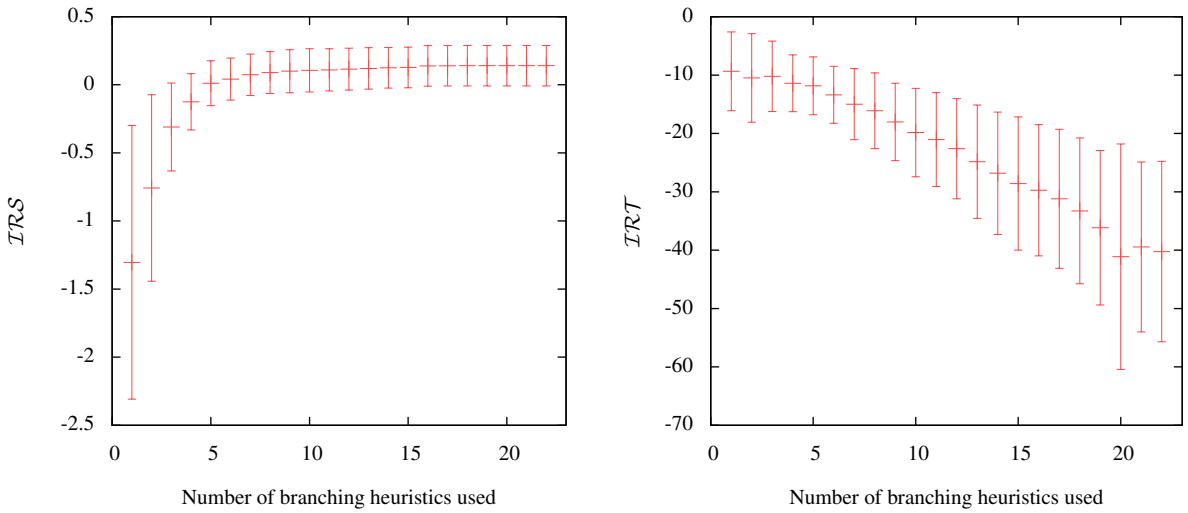
Figures 3.7 and 3.8 present the performances of the reduced oracle method for every value of c. The two versions of the method (applying or not the preprocessing treatments are considered). The methods are evaluated according to two aspects: the size of the search-tree (measured by $\mathcal{IRS}$) and the computational time (measured by $\mathcal{IRT}$). The reference method used in the measures $\mathcal{IRS}$ and $\mathcal{IRT}$ is *Best-heur*.

Remark 2. *The reduced oracle method using $c = 22$ heuristics is the oracle method, at the exception that the equalities on the quality measure are broken by giving the advantage at the branching heuristic with the better rank in the reduce oracle method.*

The evolution of the reduced oracle method regarding the number of considered branching heuristics is similar whether preprocessing treatments are used or not. $\mathcal{IRS}$ increases following a logarithmic curve when the number of branching heuristics used increases. The growth of $\mathcal{IRS}$

Rank	Without preprocessing treatment	With preprocessing treatments
1	Z2-best	Z1-best
2	Z1-best	Z2-best
3	Triang-worst	Triang-worst
4	U1-best	U1-worst
5	U1-worst	U1-best
6	U2-best	U2-best
7	U2-worst	U2-worst
8	Z1-worst	Triang-best
9	MaxRank-best	Max-worst
10	Triang-best	MaxRank-best
11	Max-worst	SumRank-best
12	Z2-worst	MinRank-worst
13	MinRank-worst	Avg-worst
14	SumRank-best	Z1-worst
15	Avg-worst	Min-best
16	Min-best	Z2-worst
17	SumRank-worst	SumRank-worst
18	Avg-best	Avg-best
19	Min-worst	MaxRank-worst
20	Max-best	Min-worst
21	MinRank-best	Max-best
22	MaxRank-worst	MinRank-best

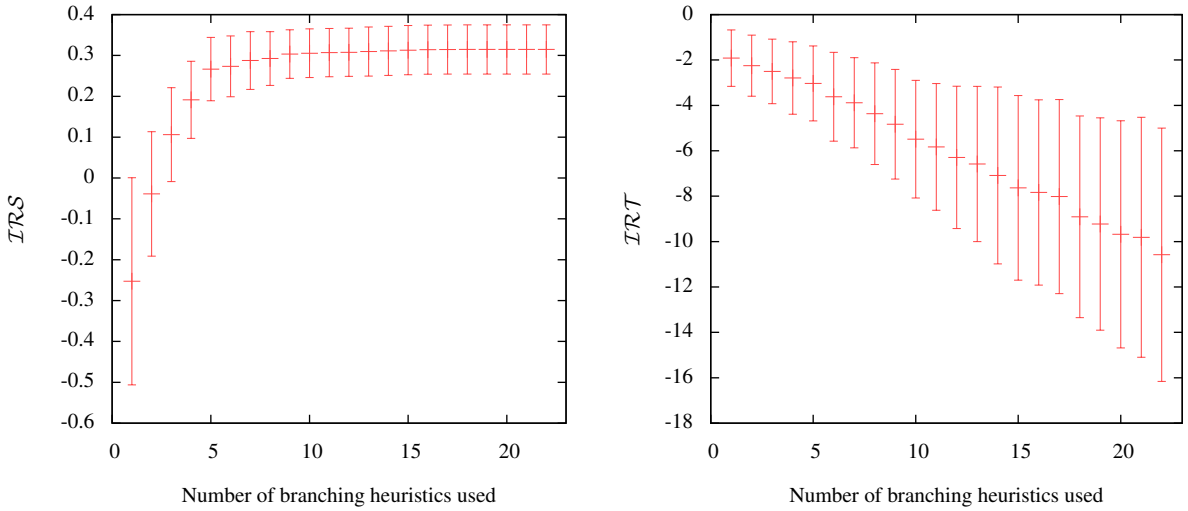
Table 3.5: Orders obtained by analyzing the performance of the branching heuristics during the oracle method, for the two versions of the method.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.7: Impact of the number of branching heuristics used in the reduced oracle method. No preprocessing treatment is applied.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.8: Impact of the number of branching heuristics used in the reduced oracle method. The preprocessing treatments are applied.

seems to stagnate when more than 5 branching heuristics are used. When the preprocessing treatments are used, the reduced oracle using 5 branching heuristics has an average $\mathcal{IRS}$ only 0.05 less than for the reduced oracle method using the 22 branching heuristics. When no preprocessing treatment is used, the difference on the average $\mathcal{IRS}$ of the reduced oracle method compared to the oracle method using the 22 branching heuristics is 0.13 when 5 branching heuristics are used and 0.05 when 9 branching heuristics are used.

$\mathcal{IRT}$ decreases linearly when the number of heuristics considered in the reduced oracle method increases. This evolution is the one expected since the number of upper bound sets computed (which is the most expensive component) is linear in the number of branching heuristics considered by the reduced oracle method.

The combination of the 5 first heuristics in the order given in Table 3.5 (with preprocessing treatments) allows to reduce the average size of the search-tree compared to a method using the same branching heuristic all along the execution. Moreover this improvement is relatively close from the one obtained by the oracle method (using 22 branching heuristics). However, the computational time required to execute this reduced oracle method is 3 times more important than when using only one branching heuristic when preprocessing treatments are used and 11 times more important when no preprocessing treatments are considered.

Those results concerning the reduced oracle method allow us to conclude that using the 5 branching heuristics *Z1-best, Z2-best, Triang-worst, U1-worst* and *U1-best* can reduce significantly the size of the search-tree (with an $\mathcal{IRS}$ of 0.26 compared to *Best-heur* when using preprocessing treatments). However, the oracle method is not suitable since the computational time is significantly more important.

In the following section, we present adaptive methods and test their use as branching strategies in our solution method for $2OKP$.

3.6 Combination of branching heuristics by adaptive methods

We aim to combine branching heuristics in a method presenting a more interesting tradeoff between the size of the obtained search-trees and the computational time than the reduced oracle method. At each iteration we have to choose over the branching heuristics the one to apply. Adaptive methods aim to choose among a list of applicable operators the one to apply at each iteration. They are generally employed in constraint programming or evolutionary algorithms. A survey on those methods can be found in (Maturana et al., 2012). The performances of the operators may vary all along the execution and the best one may not be the same for every iteration. Adaptive methods analyze the performances of the operators on the previous iterations (i.e. the empirical quality of the operators) and use this information to choose the next operator to be applied. In this manuscript, several adaptive methods are used as branching strategies to choose at each separation procedure the branching heuristic to apply. The branching heuristics are thus the operators for the adaptive methods. The algorithm is the one presented in Section 3.2 in which the branching strategy is replaced by an adaptive method.

Adaptive methods rely on quality measures for the operators applied in order to determine the empirical quality of the operator. These measures assign to each application of an operator a scalar, usually in $[0, 1]$, indicating the quality of the operator on this iteration. This value is the *reward* of the operation on this iteration. The empirical quality of an operator over several iterations is an aggregation of those successive rewards (it can for example be the average reward).

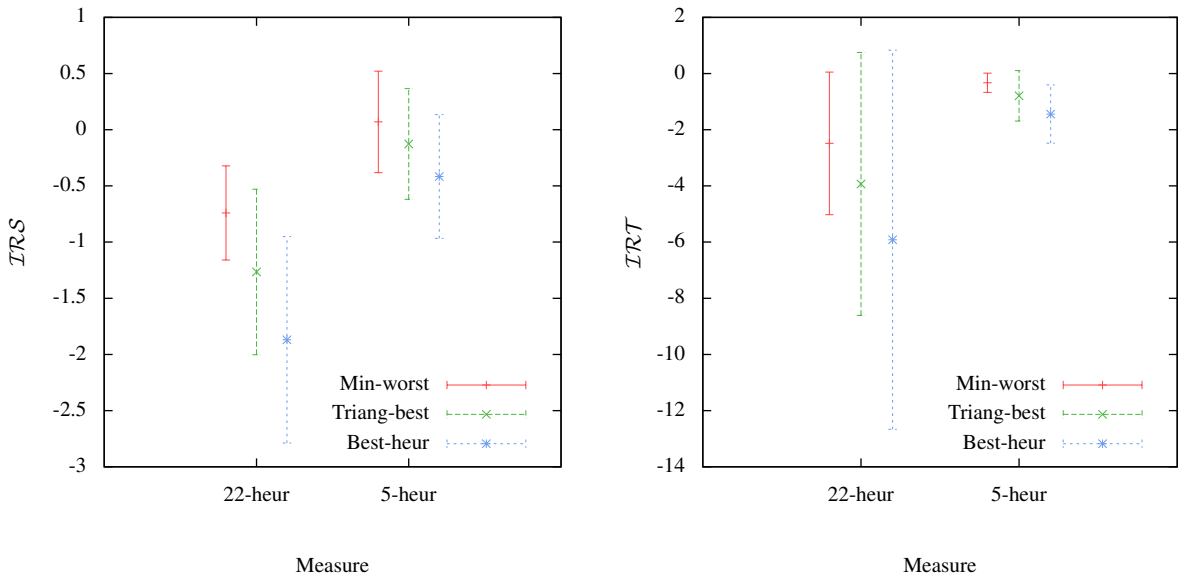
In the oracle method, the quality measure for the branching heuristics is a lexicographic order on three different criteria. Since those criteria are of different nature and are not on the same range of values, it is complex to aggregate them in a scalar value using a weighted sum. However, we have observed in Figure 3.4 that the most important evaluation criterion is *UB*. Moreover, the oracle method using only this criterion gives performances close to the best one (the difference on $\mathcal{IRS}$ is less than 0.03 when no preprocessing treatment is applied and less than 0.005 when the preprocessing treatments are applied). Thus the adaptive methods will consider the quality measure *UB* to evaluate the branching heuristics and compute the reward.

In this section, different adaptive methods are presented and tested as branching strategies. We consider both versions of the algorithm (with and without preprocessing treatments). The evaluation of the performances considers two aspects: the size of the search-tree (evaluated by $\mathcal{IRS}$) and the computational time (evaluated by $\mathcal{IRT}$). Three reference methods, applying the same branching heuristics all along the execution, will be considered: *Min-worst*, *Triang-best* and *Best-heur*. The performances of the adaptive methods are assessed on the benchmark G_1 (the smallest instances) to highlight the best value of the parameters for each method and the method leading to the best performances. Since some methods contain a random component, all the results presented here are an average over 10 executions of the algorithm.

3.6.1 Uniform wheel

Selecting randomly, at each iteration, the branching heuristic is an easy way to combine heuristics, which can be used as reference for the adaptive methods. The uniform wheel, as its name indicates, assigns to each branching heuristic the same probability to be chosen. The method can consider the 22 branching heuristics or a restricted set (as in the reduced oracle method). We have observed that the set of five branching heuristics *Z1-best, Z2-best, Triang-worst, U1-worst* and *U1-best* leads to good performances in the reduced oracle method, when considering the size of the search-trees. The results obtained when using the uniform wheel as branching strategy to choose the branching heuristics to apply are presented in Figures 3.9 and 3.10, respectively for

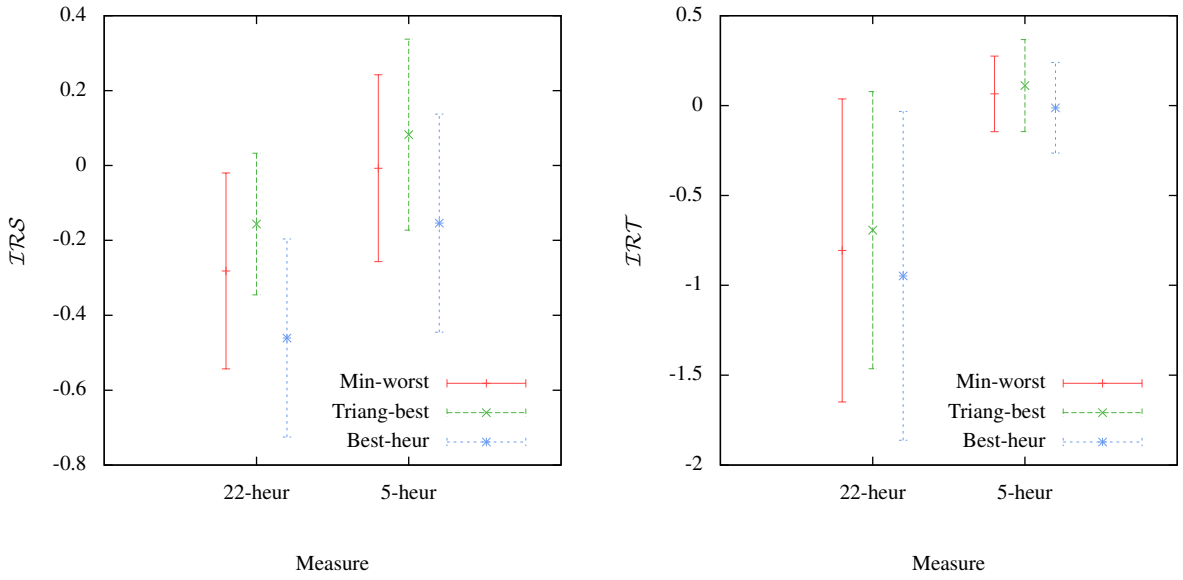
the version not applying the preprocessing treatments and applying them. *22-heur* stands for the uniform wheel using the set of 22 branching heuristics and *5-heur* for the one composed of five branching heuristics.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.9: Performances obtained when the uniform wheel is used as branching strategy when no preprocessing treatment is applied.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.10: Performances obtained when the uniform wheel is used as branching strategy when the preprocessing treatments are applied.

We can observe that the performances of the versions using the restricted set of branching heuristics offer better performances than the one using the set of 22 branching heuristics. So the observation done on the reduced oracle method seems to be confirmed here: the set of branching heuristics *Z1-best, Z2-best, Triang-worst, U1-worst* and *U1-best* are enough to bring diversity and using the 22 branching heuristics is not necessary. In the following method, the set of branching heuristics used is this set of five branching heuristics. Indeed, for all the proposed adaptive methods, the use of the restricted set offers better performances than using the 22 heuristics, since the learning is easier. In order to lighten the experiments presented in this chapter, the results obtained for the different adaptive methods using the 22 branching heuristics will not be presented in this manuscript.

When no preprocessing treatments in used, the uniform wheel using the set of five branching heuristics does not allow to improve the size of the search-tree nor the computational time compared to *Best-heur*. When compared to *Min-worst*, the size of the search-trees is slightly reduced on average.

When the preprocessing treatments are applied, the difference between the method using the uniform wheel and the reference methods are tighter. The uniform wheel does not allow to obtain better results than *Best-heur* regarding both the size of the search-tree and the computational time. The obtained computational times are slightly better than for *Min-worst* and *Triang-best*.

3.6.2 Probability matching

The probability matching, as the uniform wheel, chooses at each iteration an operator randomly. However, the probability assigned to each operator is not the same (it is not uniform) and can vary all along the algorithm. The probability matching (Goldberg, 1990) is thus a wheel-like process.

We denote $K = \{1, \ldots, k\}$ the set of operators considered (the branching heuristics in our context). The probability $P_a(t)$ of an operator $a \in K$ to be chosen at the iteration t is proportional to its empirical quality in the previous iterations, $t \in \mathbb{N}$. At each application of an operator, a reward is attributed (in our case the reward is the value of the quality measure *UB* obtained during the separation procedure). The empirical quality of an operator is the average reward obtained. We denote $R_a(t)$ the reward obtained by operator a at iteration $t \in \mathbb{N}$ if it was applied, 0 otherwise. $n_a(t)$ is the number of times the operator a has been applied up to iteration $t \in \mathbb{N}$. $Q_a(t)$ is the empirical quality of the operator a up to the iteration t, $t \in \mathbb{N}$. For the probability matching, the empirical quality of an operator is the average of the rewards: $Q_a(t) = \dfrac{\sum_{l=1}^{t} R_a(l)}{n_a(t)}$ for the operator $a \in K$ and the iteration $t \in \mathbb{N}$.

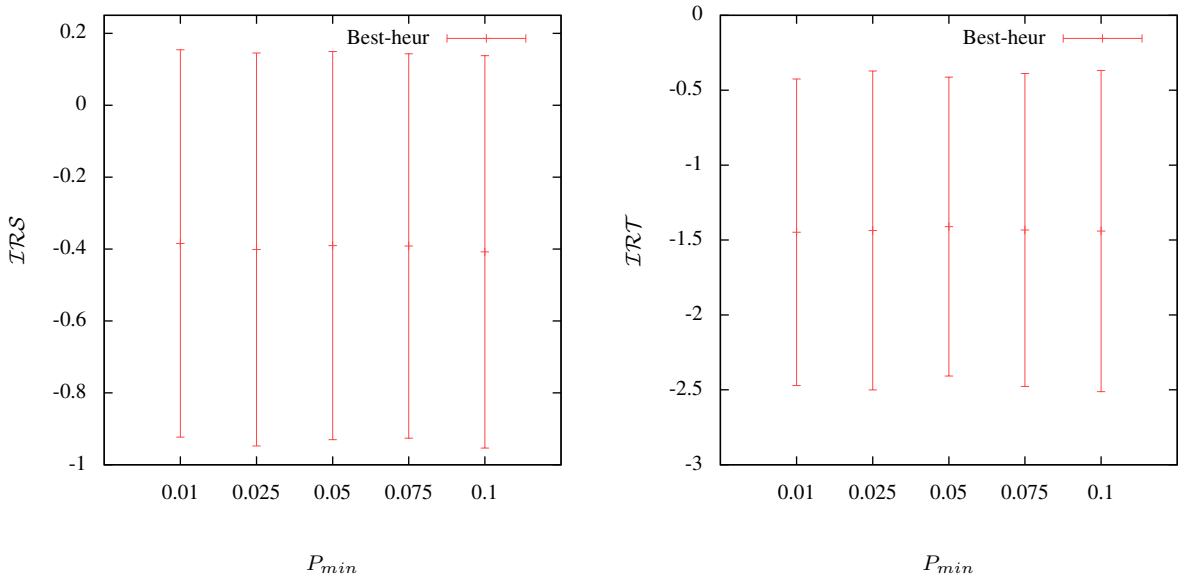
An operator can have a poor empirical quality until a given iteration, but be better than the others later in the execution. The probability that this operator has to be chosen should not be too low, so that it can still be chosen. Thus a minimum probability P_{min} is established so that no operator can have a probability lower than P_{min}. We denote P_{max} the maximum probability, $P_{max} = 1 - k\,P_{min}$. P_{min} is the only parameter of this method. If P_{min} has a too small value, some operator might never be applied and if it is too high the algorithm will often apply an other operator than the best empirically.

At the beginning of the algorithm, the probability is $\dfrac{1}{k}$ for each operator. At each iteration, an operator is randomly chosen according to the probabilities $P_a(t), a = 1, \ldots, k$. At a given

iteration $t+1, t \in \mathbb{N}$, the probability of the operator a is calculated by the formula:

$$P_a(t+1) = P_{min} + (1 + k\, P_{min}) \frac{Q_a(t)}{\sum_{a'=1}^{k} Q_{a'}(t)}$$

The probability matching is employed as branching strategy in our solution method. It considers the set of five branching heuristics (operators for the probability matching) highlighted in Section 3.5.3. The reward attributed to the branching heuristics is the quality measure *UB*. Several values of the parameter P_{min} have been tested. Since the method considers 5 heuristics, the minimum probability has to be between 0 and 0.2. In order to differentiate significantly the high and low probabilities, we did not consider P_{min} higher than 0.1. The performances obtained for the different values of the parameters are presented in Figures 3.11 and 3.12. To compare the parameters, the only reference method used here is *Best-heur* (the two other references give the same information).



(a) According to the size of the search-tree

(b) According to the computational time

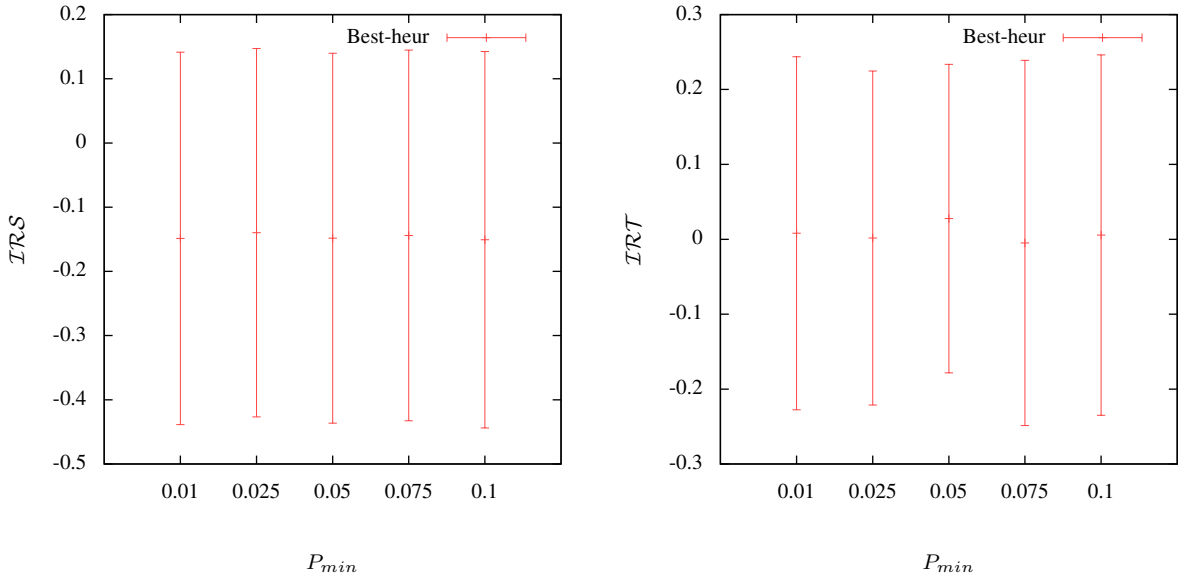
Figure 3.11: Impact of P_{min} on the solution method using the probability matching as branching strategy when no preprocessing treatment is applied.

We can see that the probability matching does not allow to improve the results obtained for *Best-heur*, neither regarding the computational time nor the size of the search-tree, for both versions of the algorithm (using or not preprocessing treatments). The influence of P_{min} is weak, but the method seems to present slightly better performances for $P_{min} = 0.05$.

The comparison with the other references (*Min-worst* and *Triang-best*) are presented for $P_{min} = 0.05$ in Figures 3.19 and 3.20, in which the performances of all the adaptive methods are presented.

3.6.3 Adaptive pursuit

Adaptive pursuit (Thierens, 2005) is an extension of the probability matching. A new parameter β is introduced controlling the rapidity with which the best operator converges to P_{max} while



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.12: Impact of P_{min} on the solution method using the probability matching as branching strategy when the preprocessing treatments are applied.

the others converge to P_{min}. β is called the learning rate. The choice of the operator to apply at an iteration is the same than in the probability matching and the notation used here is the one defined for the probability matching. The main difference with the proportional wheel is the update of the probabilities. We denote a^* the operator for which $Q_a(t)$ is the highest, $a \in \{1, \ldots, k\}$. The new probability for a^* is calculated with the formula:

$$P_{a^*}(t+1) = P_{a^*}(t) + \beta\,(P_{max} - P_{a^*}(t)).$$

The probabilities of the other operators $a \in \{1, \ldots, k\} \setminus \{a^*\}$ are calculated according to the formula:

$$P_a(t+1) = P_a(t) + \beta\,(P_{min} - P_a(t)).$$

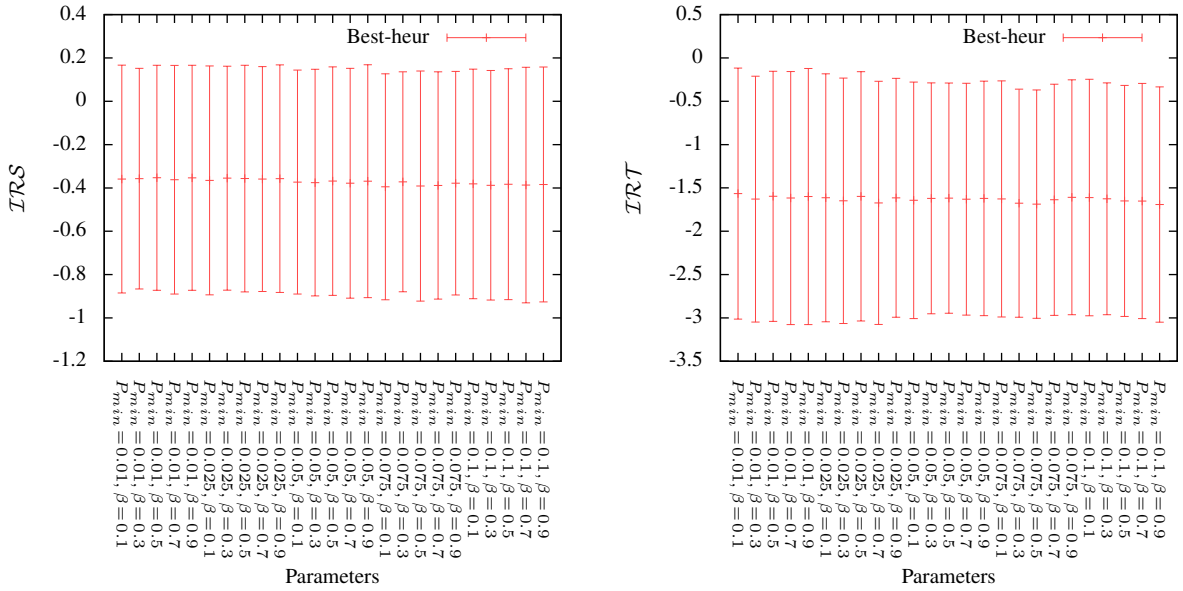
The adaptive pursuit method considers two parameters: P_{min} and β. The adaption rate β is in [0,1].

The adaptive pursuit is tested as branching strategy (to choose the branching heuristic to apply at each separation) in our method. The performances obtained are presented in Figures 3.13 and 3.14 for different values of parameters. The reference method used for $\mathcal{IRS}$ and $\mathcal{IRT}$ is *Best-heur*.

The values of the parameters do not impact strongly the performances of the algorithm. The set of parameters $P_{min} = 0.05$ and $\beta = 0.7$ seems to give the best performances. This method does not give result significantly better than the uniform wheel.

3.6.4 Upper confidence bound

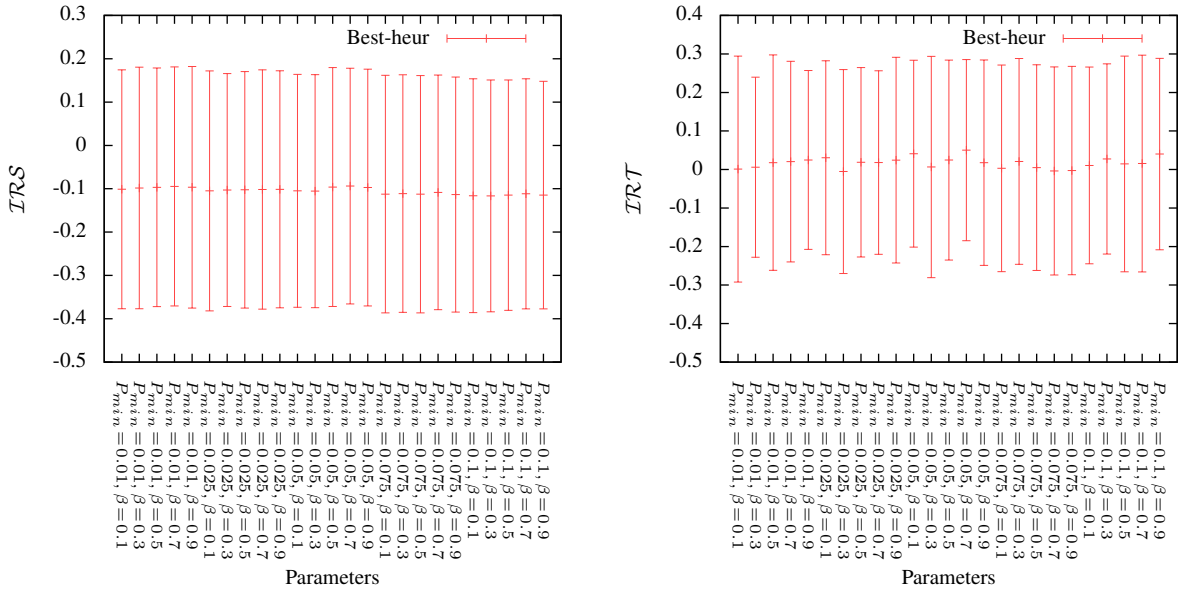
A multi-armed bandit (MAB, Berry and Fristedt (1985)) is composed of K arms. For each arm, the player has a probability p_a, $a \in \{1, \ldots, K\}$ to get a reward equal to 1 and otherwise he



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.13: Impact of the parameters on the solution method using the adaptive pursuit as branching strategy when no preprocessing treatment is applied.



(a) According to the size of the search-tree

(b) According to the computational time

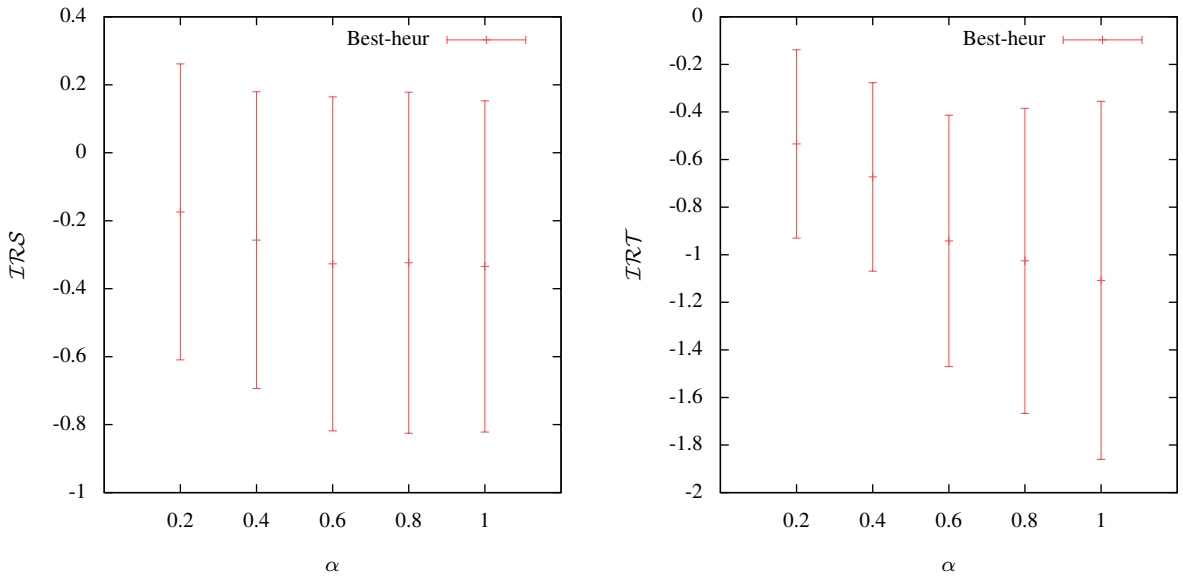
Figure 3.14: Impact of the parameters on the solution method using the adaptive pursuit as branching strategy when the preprocessing treatments are applied.

gets a reward equal to 0. The probabilities assigned to each arms are independent and unknown. The aim of the player is to minimize the regret, i.e. the difference between the maximal possible rewards and the rewards obtained.

The upper confidence bound (UCB, Auer et al. (2002)) achieve the optimal regret for MAB. In UCB the tradeoff between exploration and exploitation is considered. At each iteration, the algorithm chooses the operator a with the highest value for the formula (using the notations presented in Section 3.6.2):

$$Q_a(t) + \alpha\sqrt{\frac{2\ \log \sum_{a'=1}^{k} n_{a'}(t)}{n_a(t)}}$$

In this formula, the first part is the exploitation part, it tends to take the best operator. The second part is the exploration part, it ensures that each operator is taken infinitely many times as $t \in \mathbb{N}$ grows to infinity. $\alpha \in \mathbb{R}$ is the scaling factor balancing the exploration and the exploitation parts, which are classic in reinforcement learning. α is the only parameter of this method. $Q_a(t)$ is the empirical quality of the operator a, as defined in the section dealing with probability matching (in Section 3.6.2). UCB has many variants, see DaCosta et al. (2008) for further details.



(a) According to the size of the search-tree

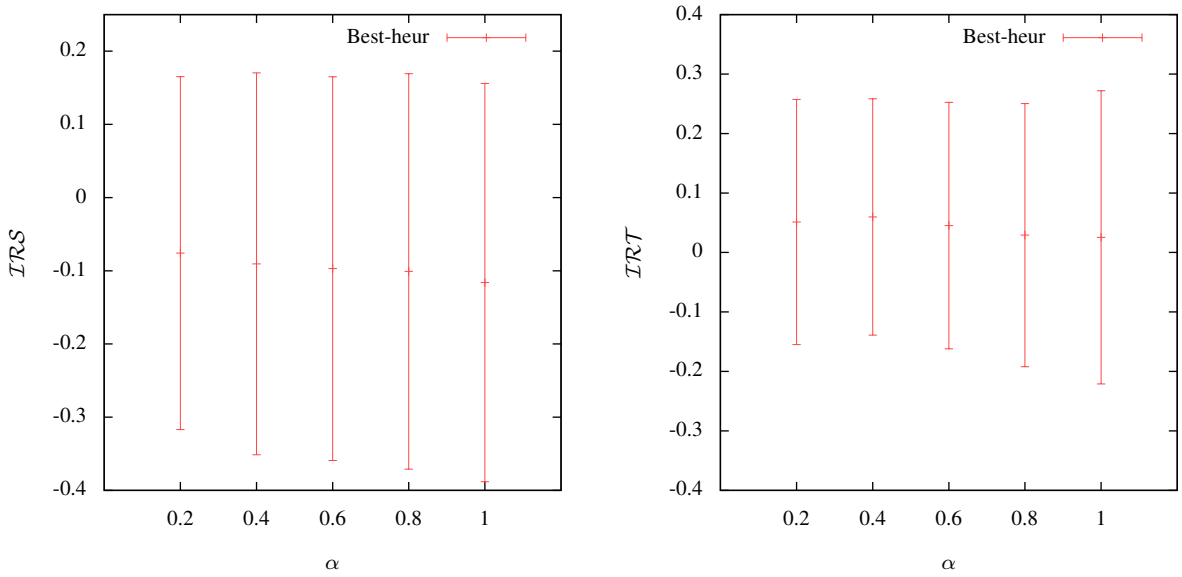
(b) According to the computational time

Figure 3.15: Impact of the value of α of the UCB when no preprocessing treatment is applied.

The performances of the algorithm using the UCB method as a branching strategy increase when the parameter α decreases. The difference is more visible when no preprocessing is used. $\alpha = 0.2$ seems to be the best value for the parameter. This value does not give the best average for $\mathcal{IRT}$ in Figure 3.16b. However, the difference between the average $\mathcal{IRT}$ for the different values of α is small and the computational times are small too. So this divergence between $\mathcal{IRS}$ and $\mathcal{IRT}$ can be explained by measure imprecision.

In this work we also considered two variants of UCB. The empirical quality of the operators varies all along the execution and in particular the best operator can deteriorate. The UCB method can take a long time before the new best operator emerges. Both variants aim to handle those changes of quality.

The first variant is the dynamic multi-armed bandit (DMAB). The Page-Hinkley test (Page, 1954) is used in the DMAB method to detect changes in the reward. The Page-Hinkley test measures the difference between the reward obtained at this iteration with the average reward obtained



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.16: Impact of the value of α of the UCB when the preprocessing treatments are applied.

for the same operator in the previous execution. If this difference exceeds a given threshold γ then the method interprets it as a change in the quality obtained by the operators and the UCB method is restarted from scratch. This restart makes it possible to find quickly the new best operator. When the value of γ is small then the method is really sensitive to changes and might restart very frequently. On the contrary when the value of γ is high, the method does not restart frequently. The DMAB considers thus two parameters: the threshold γ and the scaling factor α. Experimentally the best values for those parameters are $\alpha = 0.2$ and $\gamma = 0.8$, the result of theses experiments can be found in Appendix A.1. The performances obtained for this method are overall worst than those obtained for the classical UCB.

The second variant is the UCB with a sliding time window. The numbers of iterations, on which the empirical quality is calculated, is restricted and only considers the last iterations. The size of the window considered is a parameter of this method. In the experiments the size of the window depends on the number of variables in the considered instances. The results obtained using this method as branching strategy are presented in Appendix A.2. Same as for the DMAB, the overall results obtained for the UCB method with sliding window are worst than for the classical UCB.

Figures 3.19 and 3.20 present the performances of those three methods (only with the tuning of parameter giving the most promising results) with all the other adaptive methods and compare them to the three references (*Min-worst*, *Triang-best* and *Best-heur*).

3.6.5 Vote

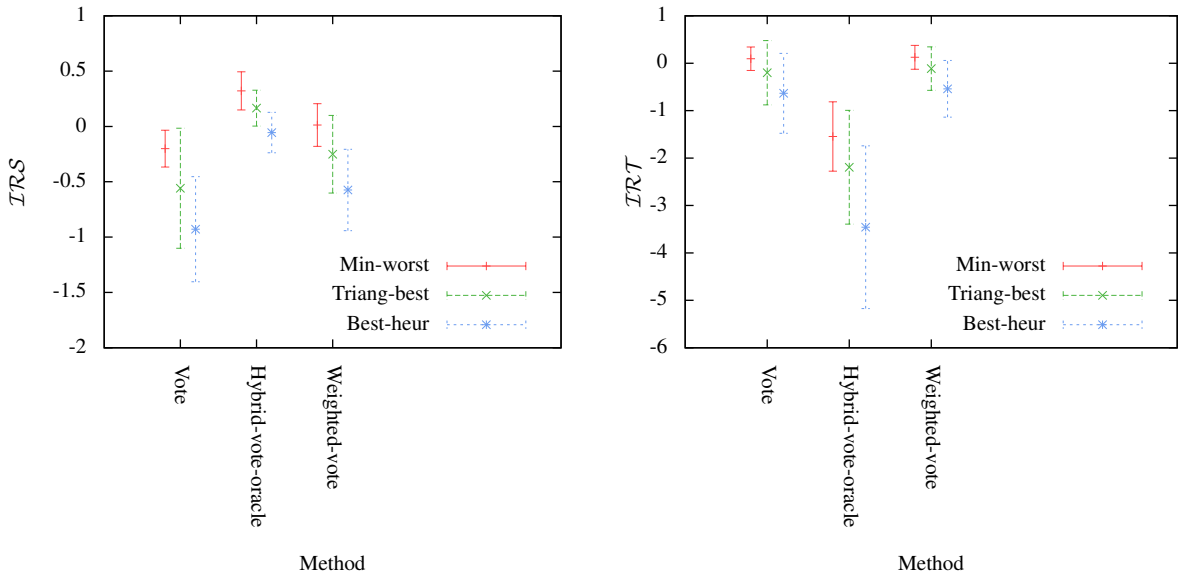
We have already mentioned that some branching heuristics can select the same variable to branch on. We can suppose that a variable selected by several branching heuristics is a good choice (it is the best according to different criteria). We elaborated a method based on this supposition, called the *vote* method. In this method, the branching variable is the one that has been selected by

the more branching heuristics, if there exist equalities they are broken randomly. This method does not have any parameter. We consider the same set of five branching heuristics used previously.

An inconvenient of the vote method as branching strategy is that at many iterations, there does not exist only one branching heuristic winning the vote, but several (even all of the heuristics if they have all selected different variables). The choice of one or another branching heuristic is then a random choice, so there is not any guarantee on the pertinence of the choice. This is the reason why we built a hybrid method mixing the vote method (which allows to choose a variable quickly) and the reduced oracle method (insuring the quality of the separations). At each separation, if there exist only one heuristic winning the vote then this heuristic is chosen for the branching, otherwise the oracle method is employed to break the equalities. This method does not have any parameter.

The last adaptive method is a variant of the hybrid method between the vote and the reduced oracle, called the *weighted vote*. The selection of the branching variable is the same than in the previous method except that the votes are weighted. At the beginning of the execution all the branching heuristics have the same weight. When the reduced oracle method is executed (when they are equalities), the weight of branching heuristics selected by the reduced oracle method increases (by one). The idea of this method is to give more importance to the branching heuristics which have been chosen by the oracle part.

The performances of the three vote oriented methods are presented in Figures 3.17 and 3.18. *Vote* denotes the first method presented in this section, *Hybrid-vote-oracle* is the hybrid method mixing the vote method and the reduced oracle method and *weighted vote* is the last method presented.

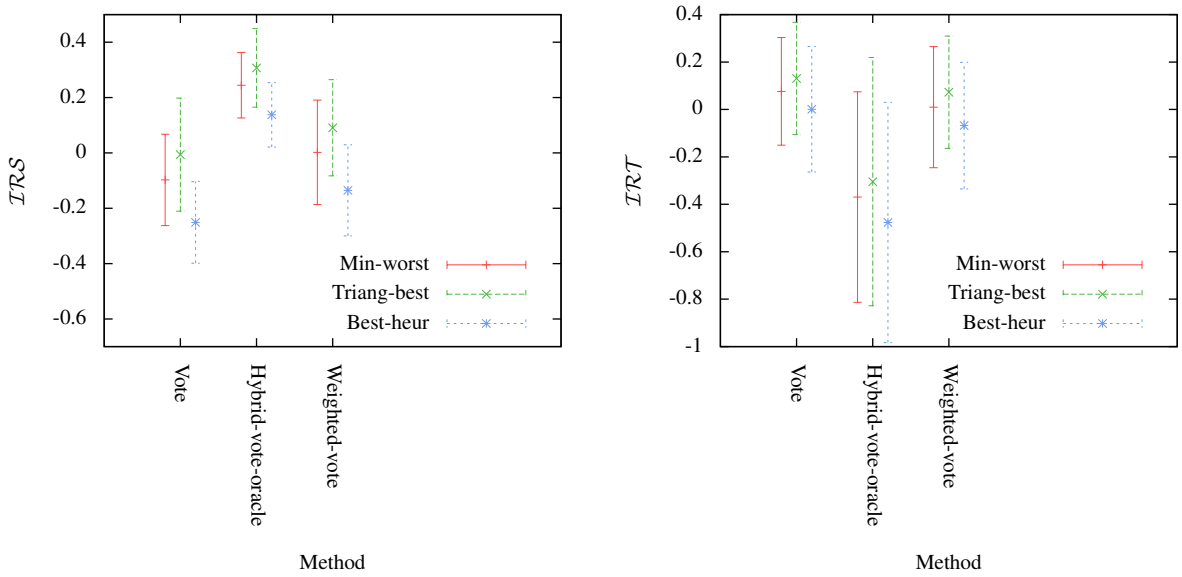


(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.17: Performances of the vote, the hybrid vote and oracle and the weighted vote method as branching strategies when no preprocessing treatment is applied.

The hybrid vote and reduced oracle method allows to reduce the significantly the size of the search-tree, however the computational time is much more important (in average four time more



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.18: Performances of the vote, the hybrid vote and oracle and the weighted vote method as branching strategies when the preprocessing treatments are applied.

important). Indeed this method applies the reduced oracle method when the vote does not allow to select one branching heuristic. The performances obtained for the hybrid method are actually similar to those obtained for the reduced oracle method (important reduction of the search-tree and increase of the computational time).

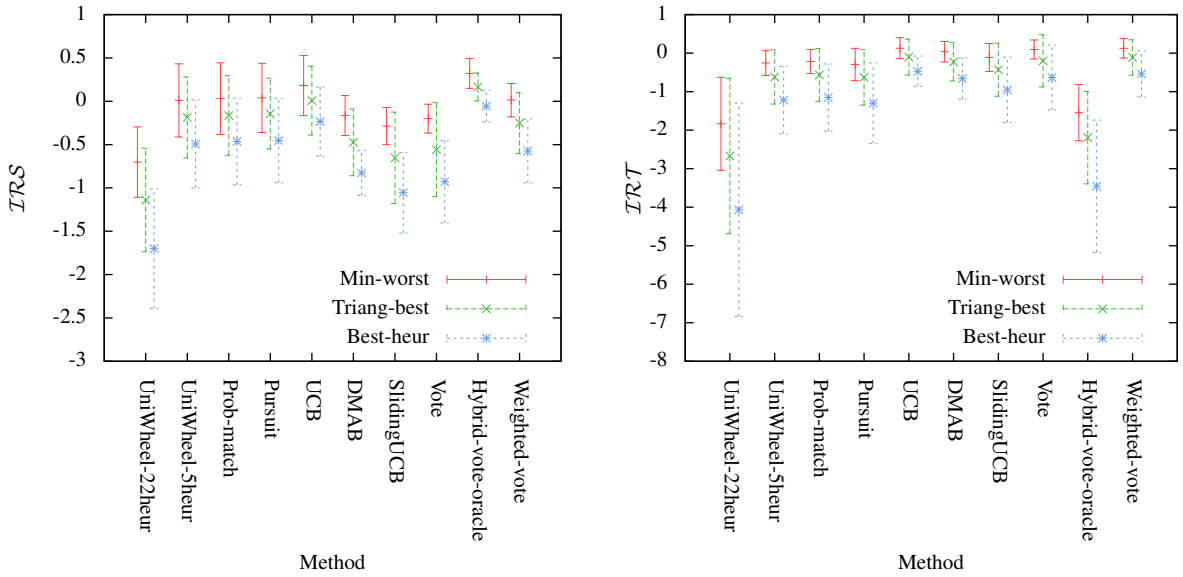
Thanks to the dynamic adaptation of the weights in the weighted vote method, less iterations of the reduced oracle method are performed. This method presents better results than both the hybrid method and the vote method, in terms of computational time and size of the search-tree.

3.6.6 Summary of the adaptive methods

Figures 3.19 and 3.20 summarize the performances obtained for all the adaptive methods presented in this section, for the instances of the group G_1 (smallest instances). The three references are considered. The name of the adaptive method has been reduced in the legend: *UniWheel* stands for the uniform wheel method (it is followed by the group of branching heuristics used), *Prob-match* stands for the probability matching method, etc.

Even if the size of the search-tree is an indicator of the performance of the method, here we are more interested in the computational times. Indeed the aim of this chapter is to elaborate a dynamic branching strategy efficient in practice to solve $2OKP$. We can notice that the use of adaptive methods makes to possible to reduce the computational time more significantly when the preprocessing treatments are used. Two method are distinctly worse than the others: the uniform wheel based on the 22 branching heuristics and the hybrid method. The performances of the others are quite close. However, only two adaptive methods have a positive $\mathcal{IRT}$ with respect to one of the reference method (*Min-worst*): the UCB and the weighted vote methods.

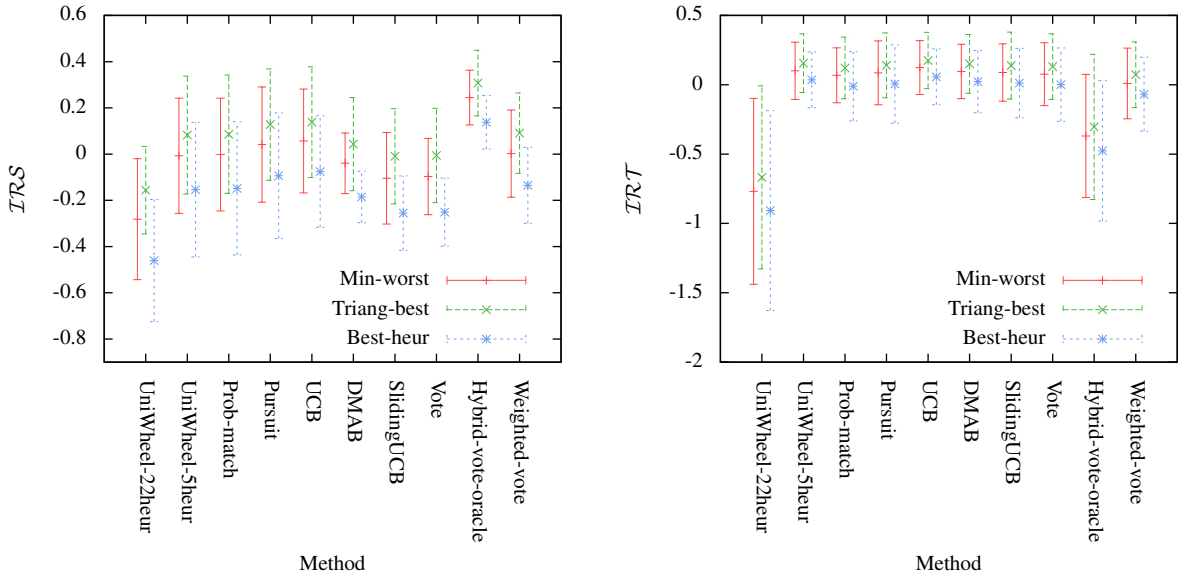
When the preprocessing treatments are used, the difference on $\mathcal{IRS}$ are more important. Except for the uniform wheel with 22 heuristics and the hybrid method, the adaptive methods allow



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.19: Performances of the adaptive methods as branching strategies when no preprocessing treatment is applied.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 3.20: Performances of the adaptive methods as branching strategies when the preprocessing treatments are applied.

to obtain a positive value of $\mathcal{IRT}$ regarding at least one reference. The adaptive method leading to the best performances in terms of computational time, over the three reference methods, is the method UCB. Table 3.6 presents the performances obtained for the UCB method, *Min-worst*,

Triang-best and *Best-heur* for all the instances of the benchmark, i.e. the group G_3. The size of the search-trees (*nbNodes*) and the computational time (expressed in seconds) are presented. Since the UCB method contains a random component, the results are presented as an average over 10 executions. Since the computational time is lower when the preprocessing treatments are executed, Table 3.6 only presents the performances obtained for the version of the algorithm using those treatments.

Strategy	UCB		Weighted vote		Min-worst		Triang-best		Best-heur	
Instance	nbNodes	time	nbNodes	time	nbNodes	time	nbNodes	time	nbNodes	time
2KP50-11	509.3	0.196	**501.1**	**0.176**	541	0.265	600	0.279	419	0.249
2KP50-50	571.3	**0.198**	**468.1**	0.207	528	0.314	660	0.322	523	0.241
2KP100-50	2593.4	1.329	**2521.5**	1.315	2665	1.228	2812	**1.200**	2513	1.376
2KP50-1A	257.7	**0.083**	**216.4**	0.092	256	0.097	312	0.138	240	0.082
2KP100-1A	**2786.4**	1.281	3112.7	1.359	3651	1.427	3226	**1.232**	3042	1.350
2KP150-1A	**3527.1**	3.717	6774.9	3.526	6850	3.496	5923	**3.054**	5923	3.099
2KP200-1A	**7563.1**	9.835	11765.5	10.657	12595	**9.569**	11968	9.851	11968	11.369
2KP250-1A	**9881.8**	23.240	19101.6	24.263	17333	**20.255**	18324	22.397	17197	19.857
2KP300-1A	**10744.3**	41.745	26784.1	43.373	29039	38.790	23414	**38.052**	21325	37.368
2KP50-1B	306.6	0.134	308.4	**0.113**	**281**	0.151	297	0.152	277	0.136
2KP100-1B	2971.5	1.865	**2886.2**	1.574	2888	**1.413**	3576	1.545	2822	1.415
2KP150-1B	**5611.1**	5.505	9662.6	5.654	10082	6.502	9693	**5.317**	9372	5.682
2KP200-1B	**5904.4**	9.153	10787.4	9.603	12617	8.962	9475	**8.041**	9475	8.602
2KP250-1B	**9290.5**	24.884	21885.1	25.807	20389	**22.163**	19594	22.189	17815	19.948
2KP300-1B	**9585.7**	35.510	23041.8	35.901	30347	36.486	20000	**30.320**	20000	31.806
2KP50-1C	1084.9	0.247	932.9	**0.213**	1161	0.443	**893**	0.318	893	0.301
2KP100-1C	1870.9	**0.578**	1968.3	0.585	**1780**	0.835	2956	0.766	1590	0.762
2KP150-1C	**3931.7**	3.139	10979.5	3.859	6037	**2.574**	9632	3.459	6037	2.960
2KP200-1C	**6686.2**	10.885	30573.7	13.946	35444	13.375	24741	**9.964**	24741	9.773
2KP250-1C	**7744.3**	30.134	38598.6	29.746	51230	31.530	29868	**21.123**	29749	20.331
2KP300-1C	**9991.6**	41.048	79942	53.027	64003	42.771	70378	**38.343**	53237	32.649
2KP50-1D	**1058.9**	0.311	1337.1	**0.259**	1118	0.369	1259	0.371	1118	0.244
2KP100-1D	1384.4	0.558	**1353.5**	0.616	1677	**0.552**	2242	0.625	1231	0.530
2KP150-1D	**5595.3**	4.687	13911.8	5.221	12929	**4.380**	17786	5.352	12929	4.562
2KP200-1D	**7359.6**	14.293	29967.3	14.945	33025	13.620	24415	**11.334**	24415	11.696
2KP250-1D	**8048.3**	**15.668**	26190.7	19.205	23357	15.866	25265	16.094	23007	15.778
2KP300-1D	**6806.3**	**9.948**	16372	11.456	15685	10.409	17142	10.604	12994	9.432
4W50W1	**2357.1**	**0.611**	2458.4	0.631	2816	0.804	2429	0.816	1921	0.604
4W100W1	**4340**	5.680	14985.3	5.737	16143	5.759	11840	**4.364**	11840	4.735
4W150W1	**6977.5**	21.515	37324.5	24.022	37150	22.664	28417	**17.882**	28417	17.679
4W200W1	**12553.6**	74.111	72742.3	77.830	81649	72.871	53591	**51.156**	53591	51.354
4W250W1	**13393.4**	164.818	112226	179.816	110386	137.549	76161	**114.847**	76161	112.597
4W300W1	**16978**	371.751	-	-	180994	341.312	109272	**242.358**	109272	239.574
F5050W01	176	0.071	**147**	0.090	199	**0.060**	153	0.085	151	0.051
F5050W02	276.3	0.104	260.8	0.112	291	0.156	**221**	**0.082**	221	0.085
F5050W03	663.1	0.193	**636.6**	**0.180**	648	0.231	681	0.287	612	0.193
F5050W04	213	0.114	**182.8**	0.104	192	**0.101**	298	0.107	180	0.099
F5050W05	1043.7	**0.220**	1011.2	0.220	1037	0.314	**956**	0.329	884	0.323
F5050W06	286.5	0.071	300.8	0.059	**270**	**0.051**	359	0.056	222	0.047
F5050W07	306.1	0.109	290.4	0.062	**227**	**0.057**	521	0.072	221	0.151
F5050W08	569.5	0.129	503.5	**0.127**	**485**	0.199	540	0.194	464	0.175
F5050W09	**816.6**	0.228	949.5	0.235	1093	0.209	1152	**0.208**	693	0.301
F5050W10	984.6	0.323	992.4	**0.273**	967	0.367	**959**	0.416	890	0.364
K5050W01	504.9	0.163	**459.2**	**0.150**	605	0.255	540	0.247	434	0.234
K5050W02	462.3	0.113	383	0.113	483	0.114	**354**	**0.107**	317	0.100
K5050W03	846.7	0.232	797	0.193	**727**	**0.188**	893	0.209	722	0.282
K5050W04	708.4	0.159	644.3	0.141	724	0.120	**609**	**0.114**	599	0.125
K5050W05	1032.5	0.172	966.7	0.154	**903**	0.132	1025	**0.132**	903	0.150
K5050W06	244.9	0.103	**194.1**	**0.086**	223	0.087	319	0.088	223	0.109
K5050W07	753.9	0.203	764.3	0.201	**628**	**0.162**	923	0.185	628	0.238
K5050W08	1180	0.325	**1099.8**	0.274	1108	**0.246**	1133	0.390	1034	0.405
K5050W09	**282**	0.111	307.2	0.106	283	**0.093**	397	0.098	256	0.091
K5050W10	1243.9	0.200	1173.7	0.254	**1008**	**0.193**	1128	0.193	979	0.180

Table 3.6: Comparison of the solution methods using *UCB*, *weighted-vote*, *Min-worst*, *Triang-best* and *Best-heur* as branching strategies.

	instance size	UCB	weighted vote
% best than Min-worst	50	51.85	55.56
	100	50.00	50.00
	150	40.00	20.00
	200	20.00	0.00
	250	40.00	20.00
	300	60.00	20.00
	global	**52.83**	**58.49**
% best than Triang-best	50	51.85	62.96
	100	33.33	33.33
	150	40.00	20.00
	200	20.00	0.00
	250	20.00	0.00
	300	20.00	0.00
	global	**60.38**	**60.38**

Table 3.7: Percentage of instances for which the adaptive methods UCB and weighted vote give a smaller computational time than the reference methods *Min-worst* and *Triang-best*, regarding the size of the instances.

We printed in bold and blue, for each instance, the smallest search-tree obtained and in bold and green the smallest computational time, we excluded *Best-heur* since it is not a realistic method. The instance *4W300W1* exceeded the memory size allocated for the adaptive method weighted vote. For some small instances, the smallest computational time can be obtained for a method leading to a search-tree that is not the smallest. This phenomenon can be explained by the fact that computational time are of the order of 0.1 seconds and that the measure of the computational time can be imprecise.

Table 3.7 presents the percentage of instances for which the adaptive methods give smaller computational time. The UCB method allows to reduce the computational time, according to the reference methods, for more than 52% of the instances and the weighted vote method for more than 58% of the instances. The performances of the methods varies with the size of the considered instances. If the performances of the UCB seems to be stable regarding the size of the instances compared to *Min-worst*, when compared to *Triang-best* they seem to slightly decrease when the size of the instance increase. The weighted vote method also gives better performances on instances of 50 variables than on bigger instances, compared to both reference methods.

3.7 Conclusion

In this chapter, we compared different branching strategies in a two phase method to solve $2OKP$ in which the second phase is a branch-and-bound method. As noticed in Chapter 2, most of the branching strategies for knapsack problems are static and based on the notion of utility.

The first part of this chapter has consisted in comparing a large number of static branching strategies (called the branching heuristics), among which some are extracted from the literature. We have remarked that the preprocessing treatments of (Jorge, 2010) and (Delort, 2011) impact significantly the performance obtained by the branching heuristics. While the branching heuristic *Triang-worst* seems to be by far better than the others when no preprocessing treatments is used, no branching heuristic plays the same role when the preprocessing treatments are applied. The

performance obtained for the branching heuristics are close to one another when the preprocessing treatments are applied. Moreover we have observed that none of the branching heuristics presented is the best for every instance of the benchmark.

The second part of the chapter aims to determine if a combination of the branching heuristics during the solving method can lead to a more efficient solution method in practice. The oracle method showed that the size of the search-tree can be reduced by up to 34% when the preprocessing treatments are employed and 14% otherwise, compared to the smallest search-tree obtained when only one branching heuristic is applied. Moreover, the average improvement of the search-tree seems to increase with the size of the considered instance. However, the oracle method is a very expensive method that cannot be used in practice, since it considers, at each separation, 22 branching strategies before to select the best one.

The reduced oracle method, by considering a restricted set of heuristics, offers a better tradeoff between the computational time and the size of the search-tree obtained than the oracle method. But this method is still considerably more time consuming than a method using the same branching heuristic all along the execution (3 times more expensive).

Several adaptive methods have been tested as branching strategies in the solution method. Some of them present interesting computational times compared to the reference method, which use the same branching heuristic all along the execution. The upper confidence bound (UCB) method seems to be the one leading to the smallest computational time in average. It leads to a smaller computational time than the reference methods in more than 50% of the instances tested. Unfortunately those performances seem to decrease when the size of the instances increases, in particular when it is compared to the reference method *Best-heur*. However, we must recall that *Best-heur* does not apply the same branching heuristics for all the instances, it applies the one leading to the smallest search-tree which is an information we do not have for a new instance. Thus this reference method is not realistic.

We have noticed that the set of available instances is restricted when dealing with a large number of variables. It would be interesting to generate new instances with a large number of variables in order to evaluate more accurately the performances of the UCB method used as branching strategies compared to the three references.

Moreover we could work to improve the practical efficiency of the UCB method by reducing the time spent to compute the reward of a branching heuristic. Indeed the reward is based on the area of a polyhedron, whose computation might be time consuming. Using another quality measure could improve the efficiency of the measure.

The use of an adaptive method could particularly be useful in the context of a more complex problem, for example dealing with a larger number of objective functions and/or a larger number of constraints. Indeed in the context of those problems, the number of defined utilities grows and thus does the number of branching strategies.

In this chapter, we focused only on the context of a branch-and-bound method embedded in a two phase method. We elaborated a dynamic branching strategy to determine the order in which the variables are considered. However, the question of the order of the variables is also of interest in other solution methods such as the dynamic programming method for example. It would be interesting to generalize the dynamic approach to those solution methods.

4

Surrogate based upper bound set for the bi-objective bi-dimensional knapsack problem

In a branch-and-bound method all the components impact the practical efficiency of the algorithm: the separation procedure, the upper and lower bound sets, the procedure allowing to find new feasible solutions and the choice of the active node. In the previous chapter, we have analyzed the impact of the branching strategy. In the literature, a key element of the practical efficiency of a branch-and-bound method in the multi-objective context seems to be the use of the convex relaxation to compute the upper bound set. Indeed this relaxation has two main advantages: its nondominated points form the tightest convex upper bound set for the original problem and its extreme nondominated points are nondominated points of the original problem. This relaxation is "easy" to compute if the single-objective version of the problem is solvable by a polynomial or pseudo-polynomial time algorithm, which is the case for $2OKP$ for example. However, its computation can be time consuming when this is not the case, for example, for $2O2DKP$.

The surrogate relaxation is generally used in solution methods for the multi-dimensional knapsack problems, both in the single-objective and multi-objective context. For $2DKP$, the algorithm from (Fréville and Plateau, 1993) gives the tightest possible bound based on the surrogate relaxation. In the multi-objective context, if several multipliers are considered for the computation of an upper bound set based on the surrogate relaxation (for example in (Gandibleux and Perederieieva, 2011)), there is no guarantee on the quality of the obtained bound set.

This chapter deals with the definition and the computation of upper bound sets, based on the surrogate relaxation, for $2O2DKP$. It introduces two upper bound sets (Section 4.1): the Optimal Surrogate Upper Bound set (OSUB), which is the tightest upper bound set we can obtain based on the surrogate relaxation, and the Optimal Convex Surrogate Upper Bound set (OCSUB), which is the tightest upper bound set we can obtain based on the convex relaxation of the surrogate relaxation. The first part of the chapter focuses on the OCSUB. Section 4.2 presents an attempt to adapt the dichotomic method to compute the OCSUB. Section 4.3 presents properties on the convex relaxation of the surrogate relaxation. Those properties are exploited to propose two exact

algorithms: an enumerative algorithm (Section 4.4) and its improved version (Section 4.5). This second algorithm results from an accurate analysis of the surrogate multipliers and the dominance relations between bound sets. Based on the improved exact algorithm, an approximated version is derived (Section 4.6). The exact and approximate algorithms are adapted to the optimal surrogate upper bound set in Section 4.7. The proposed algorithms are benchmarked using a dataset composed of three groups of numerical instances (Section 4.8). The performances are assessed thanks to a comparative analysis where exact algorithms are compared between them, the approximated algorithm is confronted to an algorithm introduced in (Gandibleux and Perederieieva, 2011).

4.1 Definition and notations

4.1.1 Surrogate relaxation

In this chapter we deal with the surrogate relaxation of $2O2DKP$. The definition of the surrogate relaxation in this context is the following:

$$\begin{array}{lll} \max & \displaystyle\sum_{j=1}^{n} c_j^k\, x_j & k = 1,2 \\ \text{s.t.} & \displaystyle u \sum_{j=1}^{n} w_{1j}\, x_j + (1-u) \sum_{j=1}^{n} w_{2j}\, x_j \leq u\,\omega_1 + (1-u)\,\omega_2 & \\ & x_j \in \{0,1\} & j = 1,\ldots,n \end{array} \qquad (2OSR(u))$$

for $u \in [0,1]$, using the same normalization as in Section 2.3.3.

The surrogate relaxation of $2O2DKP$ is an instance of $2OKP$. A distinction between feasible solutions and points of the initial problem $2O2DKP$ and those of its relaxation $2OSR(u)$ will be necessary. We will denote $Y_N(u)$ the set of nondominated points of $2OSR(u)$, and Y_N is the set of nondominated points of $2O2DKP$. This notation is extended analogically to Y_{SN}, Y_{SN1}, Y_{SN2} and Y_{NN}. $Y_N(u)$ defines an upper bound set for Y_N. Obviously, the computation of this upper bound set is practically expensive, since it is generally composed of supported and non-supported nondominated points. The set of extreme supported nondominated points $Y_{SN1}(u)$ of $2OSR(u)$ defines an upper bound set $(\operatorname{conv} Y_{SN}(u)_N)$ for $Y_N(u)$. By transitivity, it is thus also an upper bound set for Y_N. This last upper bound set has been used by Gandibleux and Perederieieva (2011). It will be called the *convex surrogate upper bound set* (CSUB) for $2OSR(u)$ and we will denote $CSUB(u) = (\operatorname{conv} Y_{SN}(u))_N$ (it can be computed by a dichotomic method on a $2OKP$). Analogously we denote *surrogate upper bound set* (SUB) the upper bound set defined for $2OSR(u)$ by $SUB(u) = Y_N(u)$.

The purpose of this chapter is to define and compute a tight upper bound set based on the surrogate relaxation for $2O2DKP$. Next section defines two upper bound sets, the *optimal surrogate upper bound set* and the *optimal convex surrogate upper bound set* based respectively on SUBs and CSUBs.

4.1.2 Optimal surrogate upper bound set and optimal convex surrogate upper bound set

As stated in Section 1.2.4 (page 28) two upper bound sets are not necessarily comparable, this is in particular true for CSUBs defined using different multipliers. Perederieieva (2011) has used

Proposition 1 (Page 32) to "merge" a number h of incomparable CSUBs, and thus to obtain a tighter upper bound set.

Proposition 4 ((Perederieieva, 2011)). *($\cap_{i=1}^{h}(CSUB(u^i) - \mathbb{R}^2_{\geqq}))_N$ is a valid upper bound set and it dominates any of the bound sets* $(\operatorname{conv} Y_{SN}(u^i))_N$ *used in the intersection.*

Obviously, considering a larger set of CSUBs in Proposition 4 allows to obtain a tighter upper bound set. Ideally, we would like to consider all possible CSUB to obtain the tightest possible upper bound set based on CSUB.

The feasible solutions of $2OSR(u), u \in [0,1]$ are binary solutions $x \in \{0,1\}^n$, where n is the number of variables in $2OSR(u)$ (by definition of the surrogate relaxation, n is also the number of variables in the original $2O2DKP$). There exist at most 2^n possible feasible solutions for a $2OSR(u)$. Each $2OSR(u)$ is defined by a subset of those solutions. Therefore, there exists a finite number of different $2OSR(u)$ and thus a finite number of different $Y_{SN1}(u) = CSUB(u)$. Therefore, Proposition 4 can be applied to merge all CSUBs. We can thus give a first generalization of the dual surrogate problem.

Definition 20. $(\cap_{u\in[0,1]}(CSUB(u) - \mathbb{R}^2_{\geqq}))_N$ *is the tightest upper bound set based on the convex surrogate relaxation and the number of multipliers to consider in order to obtain it is finite. It is denoted the* optimal convex surrogate upper bound set *(OCSUB).*

Using the same arguments, another generalization of the dual surrogate problem is possible. Using Proposition 1 in order to merge bound sets of the kind $Y_N(u)$, we can obtain the tightest upper bound set based on the surrogate relaxation.

Definition 21. $(\cap_{u\in[0,1]}(SUB(u) - \mathbb{R}^2_{\geqq}))_N$ *is the tightest upper bound set based on the surrogate relaxation and the number of multipliers to consider in order to obtain it is finite. It is denoted the* optimal surrogate upper bound set *(OSUB).*

Since the nature of a CSUB and a SUB are different, the nature of the OSUB and the OCSUB is also different. The OCSUB is a convex upper bound set, while the OSUB is an upper bound set generally non-convex. By definition, the OSUB is tighter than the OCSUB, however the computation of the OSUB is also more expensive than the computation of the OCSUB. Indeed the computation of the OSUB requires to find $Y_N(u)$ for a finite number of multipliers $u \in [0,1]$ whereas for the OCSUB only $Y_{SN1}(u)$ is required. Because of the practical difficulty to compute the OSUB, this chapter focuses principally on the computation of the OCSUB.

In the following of this chapter, we aim to design an algorithm allowing to compute the OCSUB of a $2O2DKP$. In the next section, we attempt to apply a classical dichotomic method to compute the OCSUB, applying the method presented in (Fréville and Plateau, 1993) at each iteration.

4.2 Dichotomic method

From Definition 20, we can deduce that the OCSUB verifies the following properties: $(\text{OCSUB} - \mathbb{R}^2_{\geqq})$ is convex and $(\text{OCSUB} - \mathbb{R}^2_{\geqq})$ is a polyhedron. A first natural idea for the computation of the upper bound set OCSUB is to use the existing algorithms for the solution of the single-objective case. Indeed, the following result can be used.

Proposition 5 ((Ehrgott and Gandibleux, 2007)). *Considering a relaxation $\tilde{P}$ of a MOCO problem P, $\tilde{P}_\lambda$ is a relaxation of P_λ for all $\lambda \in \mathbb{R}^2_{\geqq}$.*

Solving a surrogate relaxation (for any surrogate multiplier) of $2O2DKP_\lambda$ with $\lambda \in \mathbb{R}^2_{\geqq}$, we obtain a solution $\tilde{x}$ whose image in objective space is given by $\tilde{y} = z(\tilde{x})$. Proposition 5 implies that $Y_N \subset (\lambda^T \tilde{y} - \mathbb{R}^2_{\geqq})$. Using different directions $\lambda^1, \ldots, \lambda^k \in \mathbb{R}^2_{\geqq}$ and denoting $\tilde{y}^1, \ldots, \tilde{y}^k$ the image of the optimal solutions of surrogate relaxations of the corresponding weighted sum problems, we obtain that

$$Y_N \subset \bigcap_{i=1}^{k} ({\lambda^i}^T \tilde{y}^i - \mathbb{R}^2_{\geqq}).$$

Thus $(\bigcap_{i=1}^{k} ({\lambda^i}^T \tilde{y}^i - \mathbb{R}^2_{\geqq}))_N$ is an upper bound set for Y_N. Obviously, this upper bound set will be tighter if we solve the dual surrogate problem for each considered weighted sum problem.

As (OCSUB $-\mathbb{R}^2_{\geqq}$) is a polyhedron, there exists a finite number of direction $\lambda^1, \ldots, \lambda^l$ defining the normal to each edge of OCSUB. Therefore, we can compute OCSUB by solving exactly the dual surrogate problem associated to each of these weighted sum problems. However, we do not know these directions initially.

This problem seems similar to the computation of the convex relaxation $(\operatorname{conv} Y_{SN})_N$ of a bi-objective combinatorial optimization problem. Indeed $(\operatorname{conv} Y_{SN} - \mathbb{R}^2_{\geqq})$ is a polyhedron and its computation can be done by a dichotomic algorithm (Aneja and Nair, 1979). It seems thus very natural to apply the same algorithm with a minor modification: the exact solution of the weighted sum single-objective problem $(2DKP)$ would be replaced by the solution of the corresponding dual surrogate problem. Nevertheless, some difficulties will immediately appear (see Example 5).

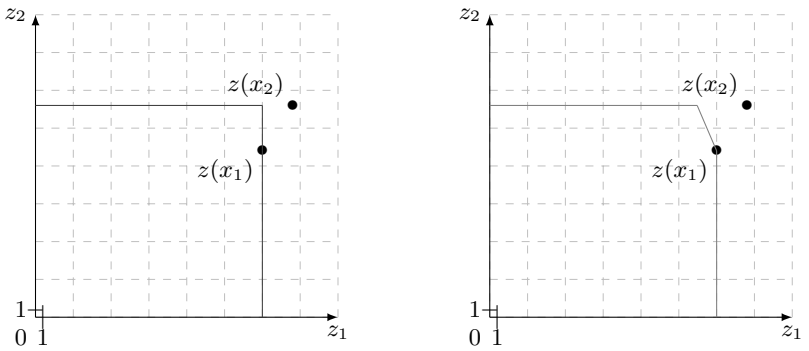
Example 5. *We consider the following instance of $2O2DKP$.*

$$\begin{array}{lrcl}
\max & 10\,x_1 + 7\,x_2 + 20\,x_3 + 7\,x_4 + 8\,x_5 & & \\
\max & 15\,x_1 + 17\,x_2 + 7\,x_3 + 4\,x_4 + 10\,x_5 & & \\
\textit{s.t.} & 3\,x_1 + 1\,x_2 + 9\,x_3 + 4\,x_4 + 9\,x_5 & \leq & 13 \\
& 13\,x_1 + 11\,x_2 + 2\,x_3 + 1\,x_4 + 7\,x_5 & \leq & 17 \\
& x_j \in \{0,1\},\ j = 1 \ldots 5 & &
\end{array} \qquad (2O2DKP\text{-}1)$$

We initialize the dichotomic method with the directions $\lambda^1 = (1,0)$ and $\lambda^2 = (0,1)$. For $\lambda^1 = (1,0)$, the optimal solution obtained solving the dual surrogate problem (Fréville and Plateau, 1993) is $x^1 = (1,0,1,0,0)$ with $z(x^1) = (30,22)$. For $\lambda^2 = (0,1)$, the optimal solution obtained solving the dual surrogate problem is $x^2 = (0,1,1,1,0)$ with $z(x^2) = (34,28)$. We can immediately note that x^2 dominates x^1. Using these two points is thus inappropriate in order to define a direction to continue the application of the dichotomy. Thus, we can only conclude that $Y_N \subset \{y \in \mathbb{R}^2 : y_1 \leq 30\} \cap \{y \in \mathbb{R}^2 : y_2 \leq 28\}$.

However, by using the direction $\lambda^3 = (3,7)$, we can improve the bound set (see Figure 4.1). Indeed, the optimal solution obtained solving the dual surrogate is then again $x^1 = (1,0,1,0,0)$ with $z(x^1) = (30,22)$. Finally, there is no immediate way to guess this direction using a dichotomic principle.

The idea behind the attempt of dichotomic method is, knowing an appropriate direction defining an edge of the OCSUB, to deduce the associated multiplier. However, Example 5 shows that these directions are not straight-forward to find. In the following, we will consider another approach based on the computation of CSUBs.



(a) Upper bound set obtained by the dichotomic method

(b) Upper bound set obtained by adding the direction (3,7)

Figure 4.1: Illustration on the example of the dichotomic method

4.3 CSUB and multiplier-set decomposition

Definition 20 shows that the computation of the OCSUB can be done by an enumeration of a finite set of CSUBs. A finite set of multipliers can therefore be used to compute these CSUBs. In this section, we study the association of multipliers to CSUBs. Ideally, we would like to obtain a multiplier-set decomposition with a one-to-one correspondence between subsets of multipliers and CSUBs. Such a decomposition would guarantee an exhaustive enumeration of all CSUB, without redundancy.

4.3.1 Critical multipliers and stability intervals

In this section, we consider individually solutions that are feasible for a problem $2OSR(u)$ where $u \in [0,1]$. To shorten the notations, this kind of solution will be called u-surrogate feasible.

The characterization of the interval of surrogate multiplier for which a given u-surrogate feasible solution x is feasible is the same as in the single-objective context, presented in Section 2.3.3 (Page 53). We recall that a u-surrogate feasible solution x can either satisfy both constraints of $2O2DKP$ or satisfies only one of them (if x does not respect any of the constraints, it does not respect this constraint neither).

If x satisfies the two constraints, then it is feasible for $2O2DKP$ and is thus u-surrogate feasible for any $u \in [0,1]$.

If x satisfies only one of the two constraints, according to Proposition 2 at page 54, we can compute $v(x)$ as

$$v(x) = \frac{\omega_2 - \sum_{j=1}^{n} w_{2j}\, x_j}{\sum_{j=1}^{n} w_{1j}\, x_j - \sum_{j=1}^{n} w_{2j}\, x_j - \omega_1 + \omega_2}$$

and x is u-surrogate feasible for any $u \in [0, v(x)]$ if x violates the first constraint and respects the second and for any $u \in [v(x), 1]$ if x respects the second constraint and violates the first. $v(x)$ is therefore a particular multiplier associated to x. Indeed, it defines one of the extremities of the interval of multipliers u such that x is u-surrogate feasible.

Definition 22.

(i) A critical multiplier u *is a multiplier such that there exists at least one solution x that is u-surrogate feasible but not $u-\epsilon$-surrogate feasible or not $u+\epsilon$-surrogate feasible, for any $\epsilon > 0$. The solution x and the critical multiplier u are called* associated.

(ii) The stability interval *of a solution x is the interval of all multipliers u such that x is u-surrogate feasible.*

Based on these observations, we can highlight three kinds of stability interval:
- $[0,1]$ for a solution feasible for $2O2DKP$;
- $[0,v(x)]$ for a solution x feasible only for the second capacity constraint of $2O2DKP$. The critical multiplier $v(x)$ is said to be of *type 0M*;
- $[v(x),1]$ for a solution x feasible only for the first capacity constraint of $2O2DKP$. The critical multiplier $v(x)$ is said to be of *type M1*.

As a critical multiplier can be associated to several solutions, it can be simultaneously 0M and M1. The x will be omitted in the notation $v(x)$, simplified in v, whenever there is no ambiguity for the considered solution x.

The definitions of critical multipliers and the different kinds of stability interval are illustrated on Example 6.

Example 6. *The $2O2DKP$ instance considered in this example is the one presented in Example 5.*

Let us consider the solution $x^1 = (1,0,1,1,0)$, its weight on the first constraint is 16 and on the second constraint it is 16. Thus the associated critical multiplier, given by the formula in Proposition 2, is $v^1 = \dfrac{17-16}{16-16-13+17} = \dfrac{1}{4}$. Since x^1 respects the second constraint and violates the first, thus x^1 is u-surrogate feasible for any $u \in \left[0, \dfrac{1}{4}\right]$ and v^1 is a critical multiplier of type 0M.

When considering the solution $x^2 = (1,1,0,0,0)$, the associated critical multiplier is $\dfrac{7}{16}$ and x^2 is u-surrogate feasible for $u \in \left[\dfrac{7}{16}, 1\right]$ and $\dfrac{7}{16}$ is a M1 critical multiplier.

When considering the solution $^3 = (0,0,1,1,1)$, the associated critical multiplier is also $\dfrac{7}{16}$ but x^3 is u-surrogate feasible for $u \in \left[0, \dfrac{7}{16}\right]$. Therefore, $\dfrac{7}{16}$ is not only a M1 critical multiplier, it is also a 0M critical multiplier.

4.3.2 Critical multipliers and CSUB

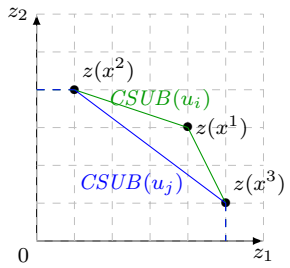
In Definition 20, the OCSUB is defined as the intersection of $CSUB(u) - \mathbb{R}^2_{\geqq}$ for a finite number of surrogate multipliers $u \in [0,1]$. Ideally all CSUB used in the intersection are different. Indeed having twice the same CSUB in the intersection does not bring any more information. It is then necessary to determine when CSUB are different. By definition $CSUB(u)$ is defined by its extreme point, which are supported nondominated point of $2OSR(u)$, for $u \in [0,1]$. Then two CSUBs are different if their sets of extreme points are not identical. This comes obviously from particular differences between the feasible sets of corresponding problems $2OSR(u)$. Lemma 2 provides a necessary condition for two CSUBs to be different.

Lemma 2. *Let $u^i \in [0,1]$ and $u^j \in [0,1]$ be two different multipliers, if $CSUB(u^i)$ and $CSUB(u^j)$ are different then at least one of the extreme supported solutions of $2OSR(u^i)$ and $2OSR(u^j)$ is feasible for one problem and not for the other one.*

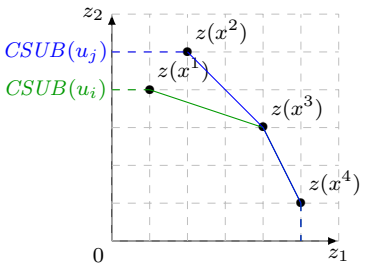
Proof. Let $x^1 \in X_{SE1}(u^i)$. Suppose that $z(x^1)$ is not an extreme point of $CSUB(u^j)$, then $x^1 \notin X_{SE1}(u^j)$. This is obviously the case if x^1 is not u^j-surrogate feasible (Figure 4.2a). Otherwise, two sub-cases need to be considered.

- There is an u^j-surrogate feasible solution x^2 dominating x^1 (Figure 4.2b). Therefore, x^2 is not u^i-surrogate feasible as it would contradict the efficiency of x^1 for $2OSR(u^i)$.
- There are two solutions $x^2, x^3 \in X_{SE1}(u^j)$ such that a point $y \in]z(x^2), z(x^3)[$ weakly dominates $z(x^1)$ (Figure 4.2c). Therefore, x^2 and/or x^3 are not u^i-surrogate feasible as it would contradict the assumption that $x^1 \in X_{SE1}(u^i)$.

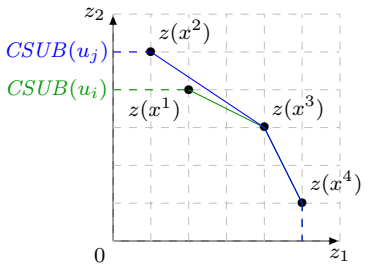
□



(a) x^1 is u^i-surrogate feasible but not u^j-surrogate feasible



(b) x^2 dominates x^1



(c) x^1 is weakly dominated by a point of the segment $(z(x^2), z(x^3))$

Figure 4.2: Illustration of the proof of Lemma 2

Remark 3. *The converse of Lemma 2 is not true. Indeed, if the solution x^1 used in the proof of this Lemma is not u^j-surrogate feasible, there may exist another solution x^2 such that x^2 is u^j-surrogate feasible and $z(x^1) = z(x^2)$.*

Consequently to Lemma 2, the differences between CSUBs come only from the (in)feasibility of extreme supported solutions of corresponding surrogate relaxations. Thus, only critical multipliers associated to these solutions will be of interest.

Definition 23. *A critical multiplier associated to a solution $x \in X_{SE1}(u)$ is also called* critical multiplier associated to $CSUB(u)$*, and x is called* solution associated to $CSUB(u)$.

Given two multipliers u and u', the proof of Lemma 2 provides an analysis of the reason why a solution x associated to $CSUB(u)$, is not necessarily associated to $CSUB(u')$: either this solution is not u'-surrogate feasible or $x \notin X_{SE1}(u')$.

Definition 24. *Let x be a solution associated to a CSUB, if x is u-surrogate feasible and $x \notin X_{SE1}(u)$ then x is called* u-masked. *More precisely, if there is a u-surrogate feasible solution x' such that x' dominates x, we will say that x is u-masked by x'. If there are two u-surrogate feasible solutions x' and x'' such that $z(x)$ is weakly dominated by a point in $]z(x'), z(x'')[$, we will say that x is u-masked by the pair of solutions x' and x''. More generally, a solution that is u-masked for all u in a set $U \subset [0, 1]$ will be called* masked on U.

Knowing the list of all multipliers associated to all possible CSUB, Proposition 6 makes it possible to find a finite set of multipliers in order to compute the set of all CSUBs.

Proposition 6. *Suppose we know the list of all multipliers associated to all CSUBs, then the list of all CSUBs is given by the following enumeration:*

(i) $CSUB(0)$ *and* $CSUB(1)$*;*

(ii) For all interval defined by two consecutive critical multipliers : $CSUB(u)$ *for a multiplier* u *in the relative interior of this interval;*

(iii) $CSUB(u)$ *for all multiplier* u *that is simultaneously 0M and M1.*

Proof. Lemma 2 implies that $CSUB(u)$ is identical for all $u \in]v^1, v^2[$ where v^1 and v^2 are consecutive critical multipliers. It is thus useless to consider two multipliers in such an open interval $]v^1, v^2[$. On the contrary, $CSUB(u)$ and $CSUB(u')$ such that u and u' belong to the interior of different intervals given by consecutive critical multipliers, implies that $CSUB(u)$ and $CSUB(u')$ may be different (but are not necessarily as the converse of Lemma 2 is not true). Knowing $CSUB(u)$ for u multiplier in the interior of all interval defined by consecutive critical multipliers is thus necessary to guarantee an exhaustive enumeration of all CSUBs (item (ii)). If 0 and 1 are not critical multiplier, one multiplier must also be considered in the interval $[0, u^{\min}[$ and $]u^{\max}, 1]$ where $u^{\min}$ and $u^{\max}$ are respectively the smallest and the greatest critical multiplier, $CSUB(0)$ and $CSUB(1)$ are appropriate choices. Nevertheless, it is not sufficient to obtain an exhaustive enumeration of CSUBs. Indeed if we consider a multiplier v that is simultaneously 0M and M1, and a small number $\epsilon > 0$ such that there is no other critical multiplier in $[v - \epsilon, v + \epsilon]$. As v is 0M then $CSUB(v)$ may be different to $CSUB(v + \epsilon)$, and as v is M1 then $CSUB(v)$ may be different to $CSUB(v - \epsilon)$. The computation of $CSUB(v)$ is thus also necessary (item (iii)). For the same reason, the computation of $CSUB(0)$ (respectively $CSUB(1)$) is also required if 0 is a 0M multiplier (respectively 1 is a M1 multiplier). Computing $CSUB(0)$ and $CSUB(1)$ is therefore required in all cases (item (i)). □

The multiplier set-decomposition described by Proposition 6 does not exactly give a one-to-one correspondence between subsets of multipliers and CSUBs (as the converse of Lemma 2 is not true) but guarantees the determination of all CSUB. However, a way to obtain the set of all critical multipliers is necessary in order to use this last statement. Proposition 7 is based on an assumption that is easier to verify: only a subset of CSUBs with their associated critical multipliers are known. In this context, there may exist unknown solutions associated to unknown CSUBs, that are either u-masked or not u-surrogate feasible for each multiplier u considered so far. In order to determine these unknown solutions, we must find multipliers u such that these solutions are potentially u-surrogate feasible and not u-masked. A characterization of the intervals of multipliers S on which a solution is masked is provided by Proposition 7.

Proposition 7. *Suppose we know some CSUBs, with their associated solutions and critical multipliers. Let* x *be an unknown solution associated to a CSUB, then* x *is* u*-masked for all* $u \in U$ *where* U *is an union of intervals. The possible patterns for these intervals, relies on the stability interval of* x*.*

(1) If the stability interval of x *is* $[0, 1]$*, then these patterns are:*

(i) $[0, v^0]$ *where* v^0 *is a known 0M multiplier;*

(ii) $[v^1, 1]$ *where* v^1 *is a M1 multiplier ;*

(iii) $[v^1, v^0]$ *where* v^0 *and* v^1 *are respectively known 0M and M1 multipliers.*

(2) If the stability interval of x *is* $[0, v]$*, then these patterns are:*

(i) $[0, v^0]$ *where* v^0 *is a known 0M multiplier such that* $v^0 < v$*;*

(ii) $[v^1, v]$ *where* v^1 *is a known 0M multiplier such that* $v^1 \leq v$*;*

(iii) $[v^1, v^0]$ *where* v^0 *(such that* $v^0 \leq v$*) and* v^1 *are respectively known 0M and M1 multipliers.*

(3) If the stability interval of x is $[v, 1]$:

(i) $[v, v^0]$ where v^0 is a known 0M multiplier such that $v \leq v^0$;

(ii) $[v^1, 1]$ where v^1 is a known M1 multiplier such that $v < v^1$;

(iii) $[v^1, v^0]$ where v^0 and v^1 (such that $v \leq v^1$) are respectively a 0M and M1 critical multipliers.

Proof.

(1) We assume that the stability interval of x is $[0, 1]$.
We suppose first that x is u'-masked by one solution x' for a given multiplier u', then x is u-masked for all multiplier u such that x' is u-surrogate feasible. In other words, if we denote by S the stability interval of x', then x is masked on S. The interval S is either of the kind $[0, v^0]$ (pattern (i)) or $[v^1, 1]$ (pattern (ii)) where v^0 and v^1 are respectively 0M and M1 multipliers. The case $S = [0, 1]$ is not possible since x is a solution associated to a CSUB.
We suppose next that x is u'-masked by the pair of solutions x' and x'' for a given multiplier u', then x is u-masked for all multipliers u such that both solutions are u-surrogate feasible. In other words, if we denote the stability interval of both solutions by S_1 and S_2 then x is masked on $S_1 \cap S_2$. If both solutions are associated to 0M critical multipliers v^{01}, v^{02} then x is masked on $[0, \min(v^{01}, v^{02})]$ (pattern (i)). If both solutions are associated to M1 critical multipliers v^{11}, v^{12}, then x is masked on $[\max(v^{11}, v^{12}), 1]$ (pattern (ii)). If one solution is associated to a 0M critical multiplier v^0 and the other one is associated to a M1 critical multiplier v^1, then x is masked on $[v^1, v^0]$ (pattern (iii)).

(2) We assume that the stability interval of x is $[0, v]$ where u is a 0M critical multiplier. In order to find the possible pattern for this second case, we just need to compute the intersection of $[0, v]$ with the pattern enumerated in the case (1). For the pattern $(1)(i)$, we necessarily have $[0, v^0] \cap [0, v] = [0, v^0]$ with $v > v^0$ (pattern (i)), otherwise x is u-masked for all u in its stability interval. For the pattern $(1)(ii)$, we either have $[v^1, 1] \cap [0, v] = [v^1, v]$ with $v \geq v^1$ (pattern (ii)) or $[v^1, 1] \cap [0, v] = \emptyset$ if $v < v^1$. For the pattern (iii), we have either $[v^1, v^0] \cap [0, v] = [v^1, v^0]$ (pattern (iii)) with $v^1 < v^0 \leq v$, or $[v^1, v^0] \cap [0, v] = [v^1, v]$ (pattern (ii)) with $v^1 \leq v < v^0$ (we always have $v^1 \leq v$ since solutions associated to v^0 and v^1 are a pair masking x).

(3) The proof is symmetric to the case (2).

□

Remark 4. *Given a unknown solution x associated to a CSUB, the (disjoint) union of intervals U on which x is masked by known solutions, is composed of zero or one interval of type $[0, v^0]$ (or $[v, v^0]$ for the case (3)), zero or one interval of type $[v^1, 1]$ (or $[v^1, v]$ for the case (2)) and zero or several disjoint intervals of type $[v^1, v^0]$.*

The exact computation of the union of intervals U would require to know the solution x. Nevertheless, Proposition 7 only gives the patterns of intervals on which an unknown x is masked. However, as the boundaries of these intervals (except v) are defined by known multipliers, it will be possible to deduce particular multipliers u such that potential unknown solutions associated to CSUBs are not u-masked by known solutions or pairs of solutions.

4.4 Total enumerative algorithm

In this section, we describe a first enumerative algorithm for the computation of the OCSUB, based on Propositions 6 and 7. The main idea of this algorithm is to determine first the set of all

multipliers associated to all possible CSUB (line 2 of Algorithm 2), and next to deduce the set of all CSUBs (line 3 of Algorithm 2). The OCSUB can finally be computed by an application of Definition 20.

Algorithm 2: Algorithm `TotalEnumerative`

```
   input : A 2O2DKP Instance
   output: bound: the OCSUB for the instance of 2O2DKP
1  begin
       /* allMult denotes the set of all enumerated multipliers        */
       /* bound denotes the bound obtained by merging CSUBs            */
2      EnumerateCriticalMultiplier(allMult ↑, bound ↑)
3      ComputeOCSUB(allMult ↓, bound ↕)
```

4.4.1 First part: the enumeration of all critical multipliers associated to all possible CSUB

In order to initialize our enumeration, we start by computing $CSUB(0)$ and $CSUB(1)$ and deducing their associated critical multipliers (line 2 of Algorithm 3). The computation of these CSUBs is indeed required in all cases according to Proposition 6. Another interesting property about these two multipliers is given by Lemma 3.

Lemma 3. *All solutions associated to any CSUB are either* 0*-surrogate feasible or* 1*-surrogate feasible or both.*

Proof. We already know that the stability interval of solutions associated to CSUBs are of three kinds: $[0, u]$, $[u, 1]$, $[0, 1]$. The statement follows. □

Consequently to this lemma, the union of intervals on which an unknown solution x (associated to an unknown CSUB) is masked by a known solution cannot be empty after this initialization.

According to Lemma 2, a way to enumerate new CSUBs is to consider multipliers u such that at least one solution associated to a known CSUB is no longer u-surrogate feasible. In other words, we need to consider multipliers u outside of the stability interval of known solutions. There will be two cases to consider, a solution is associated to a 0M multiplier or a M1 multiplier (there is nothing to do in the case of a solution for which the stability interval is $[0, 1]$ as it is u-surrogate feasible for all $u \in [0, 1]$). In the case of a 0M multiplier v^0, any multiplier $u > v^0$ will guarantee that the solution(s) associated to v^0 is (are) no longer u-surrogate feasible. Symmetrically, any multiplier $u < v^1$ will guarantee that the solution(s) associated to a M1 multiplier v^1 is (are) no longer u-surrogate feasible.

In an iteration of the procedure (line 5-10 of Algorithm 3), a critical multiplier v is considered individually (a multiplier simultaneously 0M and M1 is here considered as two different multipliers) to define a multiplier u whose value is as near as possible to v. Ideally, there must not be any critical multiplier between v and u. Thus, appropriate values u are defined by $u := v + \epsilon$ if u is a 0M multiplier, and $u := v - \epsilon$ if u is a M1 multiplier, where $\epsilon > 0$ is a small value (see Appendix B.1 for an upper bound on ϵ). $CSUB(u)$ is next computed; and its associated multipliers are deduced.

All CSUBs computed during the execution of this algorithm are merged using Proposition 4 (line 4,10 of Algorithm 3).

Proposition 8. *Algorithm 3 finds all critical multipliers associated to all possible CSUB.*

Algorithm 3: Procedure `EnumerateCriticalMultiplier`

input : A $2O2DKP$ instance
output: allMult: the list of all critical multipliers associated to all possible CSUB
bound: An upper bound set for Y_N

1 **begin**
  `/* compAllCM computes all critical multipliers associated to a CSUB */`
  `/* multToDo denotes the multipliers from which deriving a new multiplier */`
  `/* Here, a 0M and M1 multiplier is considered as two distinct multipliers */`
2 allMult $\leftarrow$ `compAllCM`$(CSUB(0)\downarrow) \cup$ `compAllCM`$(CSUB(1)\downarrow)$
3 multToDo $\leftarrow$ allMult
4 bound $\leftarrow [(CSUB(0) - \mathbb{R}^2_{\geqq}) \cap (CSUB(1) - \mathbb{R}^2_{\geqq})]_N$
5 **while** multToDo $\neq \emptyset$ **do**
    `/* Select a multiplier into the list */`
6    $v \leftarrow$ `getMult`(multToDo $\downarrow$); multToDo $\leftarrow$ multToDo $\setminus \{v\}$
7   **if** *v is 0M* **then** $u \leftarrow v + \epsilon$ **else** $u \leftarrow v - \epsilon$
8   multToDo $\leftarrow$ multToDo $\cup$ [`compAllCM`$(CSUB(u)\downarrow) \setminus$ allMult]
9   allMult $\leftarrow$ allMult $\cup$ `compAllCM`$(CSUB(u)\downarrow)$
10   bound $\leftarrow [(\text{bound} - \mathbb{R}^2_{\geqq}) \cap (CSUB(u) - \mathbb{R}^2_{\geqq})]_N$

Proof. Let u be a critical multiplier associated to a CSUB. We denote by x a solution associated to u. If u is associated to $CSUB(0)$ or $CSUB(1)$, then u is found during the initialization (Line 2 of Algorithm 3). If u is not associated to either $CSUB(0)$ nor $CSUB(1)$, then we show that u is found in an iteration of Algorithm 3. We consider thus an iteration of this algorithm and suppose that u has not been found in a preceding iteration. Proposition 7 states the pattern of intervals U the union on which x is masked by known solutions. According to Lemma 3, there is at least one interval U defined by Proposition 7 on which x is masked. The interval U has either a left-extremity defined by a M1 multiplier v^1 or a right-extremity defined by a 0M multiplier v^0 or both (see Remark 4). Algorithm 3 considers the computation of all CSUBs of the following kind: $CSUB(v^0 + \epsilon)$, $CSUB(v^1 - \epsilon)$, with possible new solutions and critical multipliers found. There are two possible conclusions: u is one of these new critical multipliers or not (i.e. all solutions associated to u are still masked). In the latter case, new critical multipliers allow to performs new iterations. Again, u will be found or not. By repeated computations of CSUBs, u will be finally found in a finite number of computations of CSUB, as the number of critical multipliers is finite and as multipliers v such that x is not v-masked by known solution(s) can always be deduced. □

4.4.2 Second part: computation of the OCSUB

After the first part, all critical multipliers associated to all CSUBs are known. Thus, Proposition 6 can be used in order to compute all CSUBs. We summarize the CSUBs computed in the part 1:

(i) $CSUB(0)$ and $CSUB(1)$ in the initialization;

(ii) $CSUB(u + \epsilon)$ for all u 0M multiplier;

(iii) $CSUB(u - \epsilon)$ for all u M1 multiplier.

Items (ii) and (iii) of the enumeration above imply that at least one multiplier has been used in the interior of intervals defined by consecutive critical multipliers such that the left hand side is 0M or the right hand side is M1. To complete Item (ii) of Proposition 6, it remains thus to use one multiplier in each interval for which the left side is M1 and the right side of which is 0M. If we denote by u the left hand side of this last kind of interval, the multiplier $u+\epsilon$ is an appropriate choice for the computation of a CSUB.

In conclusion, to achieve the enumeration of all CSUBs, it remains to compute:
- $CSUB(u+\epsilon)$ for all interval of consecutive critical multipliers $[u, u^0]$ such that u is M1 and u^0 is 0M (Item (ii) of Proposition 6);
- $CSUB(u)$ for all multiplier u that is 0M and M1 (Item (iii) of Proposition 6).

Algorithm 4 summarizes this final computation.

Algorithm 4: Procedure `ComputeOCSUB`

input : A $2O2DKP$ instance, the set of all critical multipliers allMult,
bound: an upper bound set for Y_N
output: bound: the OCSUB for the $2O2DKP$ instance

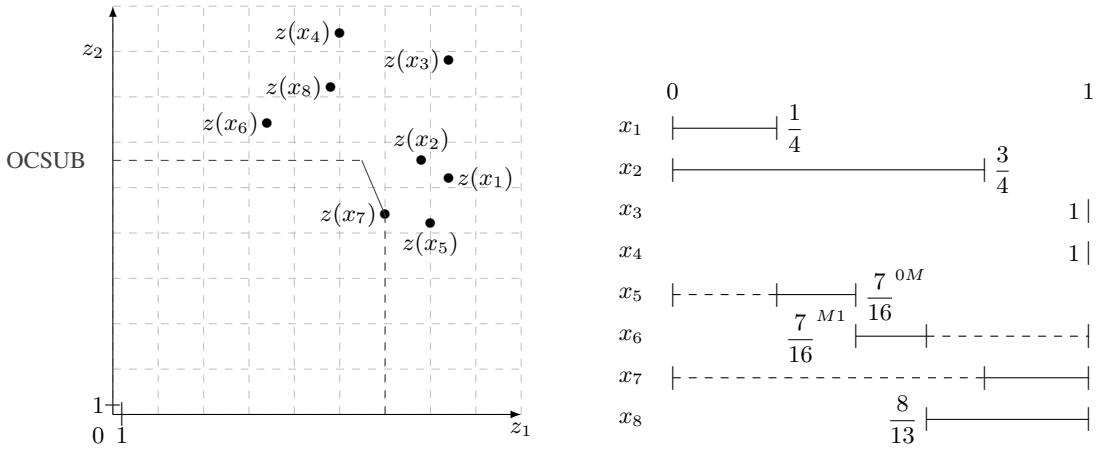
```
1 begin
2     foreach u ∈ allMult do
3         if u is 0M and M1 then bound ← [(bound − ℝ²≧) ∩ (CSUB(u) − ℝ²≧)]_N
4     foreach Interval of consecutive critical multipliers [u, v] such that u is M1 and v is 0M do
5         bound ← [(bound − ℝ²≧) ∩ (CSUB(u + ε) − ℝ²≧)]_N
```

Example 7. *The* TotalEnumerative *algorithm is performed on the instance of $2O2DKP$ introduced in Example 5. To facilitate the understanding of the output, Table 4.1 reports all solutions associated to at least one CSUB and their stability interval.*

Table 4.1: Information relative to the solutions associated to at least one CSUB

Name	Solution	Objective value	Stability interval
x_1	(1,0,1,1,0)	(37,26)	$\left[0, \frac{1}{4}\right]$
x_2	(0,1,1,1,0)	(34,28)	$\left[0, \frac{3}{4}\right]$
x_3	(1,1,1,0,0)	(37,39)	$[1, 1]$
x_4	(1,1,0,0,1)	(25,42)	$[1, 1]$
x_5	(0,0,1,1,1)	(35,21)	$\left[0, \frac{7}{16}\right]$
x_6	(1,1,0,0,0)	(17,32)	$\left[\frac{7}{16}, 1\right]$
x_7	(1,0,1,0,0)	(30,22)	$[0, 1]$
x_8	(1,1,0,1,0)	(24,36)	$\left[\frac{8}{13}, 1\right]$

The solutions are represented in objective space on Figure 4.3a. The stability intervals of those solutions are represented on Figure 4.3b. On the representation of the stability interval of a solution x_i, the dashed parts correspond to the interval of multipliers on which x_i is masked.



(a) The solutions in the objective space

(b) Stability interval of the solutions

Figure 4.3: Information relative to the eight solutions associated to a CSUB for the instance of $2O2DKP$ in Example 7

The execution of the algorithm is presented in Table 4.2.

	Nb CSUB	u	type	e	$CSUB(u+e)$	multToDo	multipliers
Initialization	1	0	-	0	$\{x_1, x_2\}$	$\mathbf{\frac{1}{4}}, \mathbf{\frac{3}{4}}$	$\mathbf{\frac{1}{4}}, \mathbf{\frac{3}{4}}$
	2	1	-	0	$\{x_3, x_4\}$	$\frac{1}{4}, \frac{3}{4}, \mathbf{1}$	$\frac{1}{4}, \frac{3}{4}, \mathbf{1}$
First part	3	$\frac{1}{4}$	0M	$+\epsilon$	$\{x_2, x_5\}$	$\mathbf{\frac{7}{16}}^{\mathbf{0M}}, \frac{3}{4}, 1$	$\frac{1}{4}, \mathbf{\frac{7}{16}}, \frac{3}{4}, 1$
	4	$\frac{7}{16}^{0M}$	0M	$+\epsilon$	$\{x_2, x_6\}$	$\mathbf{\frac{7}{16}}^{\mathbf{M1}}, \frac{3}{4}, 1$	$\frac{1}{4}, \mathbf{\frac{7}{16}}, \frac{3}{4}, 1$
	5	$\frac{7}{16}^{M1}$	M1	$-\epsilon$	$\{x_2, x_5\}$	$\frac{3}{4}, 1$	$\frac{1}{4}, \frac{7}{16}, \frac{3}{4}, 1$
	6	$\frac{3}{4}$	0M	$+\epsilon$	$\{x_7, x_8\}$	$\mathbf{\frac{8}{13}}, 1$	$\frac{1}{4}, \frac{7}{16}, \mathbf{\frac{8}{13}}, \frac{3}{4}, 1$
	7	$\frac{8}{13}$	M1	$-\epsilon$	$\{x_2, x_6\}$	1	$\frac{1}{4}, \frac{7}{16}, \frac{8}{13}, \frac{3}{4}, 1$
	8	1	M1	$-\epsilon$	$\{x_7, x_8\}$	$\emptyset$	$\frac{1}{4}, \frac{7}{16}, \frac{8}{13}, \frac{3}{4}, 1$
Second part	9	$\frac{7}{16}$	0M&M1	0	$\{x_2, x_5, x_6\}$	-	"
	10	$\frac{8}{13}$	M1	$+\epsilon$	$\{x_2, x_8\}$	-	"

Table 4.2: Example of execution of Algorithm 2. The bold values correspond to new multiplier (found during this iteration). NB: the value of $u = \frac{7}{16}$ corresponds simultaneously to one 0M and one M1. To avoid any confusion, the type of origin which is derived the value is mentioned in superscript.

The OCSUB is defined by the set of extreme points (rounded to 10^{-2}) {(30,22); (27.43,28)}.

Remark 5. *When "merging" upper bound sets based on the surrogate relaxation, using Propositions 1 (Page 32), the extreme points of the resulting upper bound set may not be feasible for any surrogate relaxation. In particular, intersection points may not have integer coordinates (as in Example 7).*

In checking Figure 4.3a and Table 4.2, a number of obtained CSUBs, outcame of the TotalEnumerative algorithm, appears dominated by other CSUBs. Such a situation may appear in general, and these dominated CSUBs are therefore useless for the computation of the OCSUB. The next section gives an analysis of the dominance relation between the different CSUBs.

4.5 0M-M1 intervals

4.5.1 Bound sets and dominance

The dominance between CSUBs is based on the following Lemma, illustrated by Figure 4.2 (in Section 4.3.2 Page 99).

Lemma 4. *We consider $CSUB(u^i)$ and $CSUB(u^j)$ two different CSUBs, defined respectively by the set of extreme points P^i and P^j :*

1. *If $P^j \subset P^i$ then $CSUB(u^j)$ dominates $CSUB(u^i)$ (Figure 4.2a)*
2. *If $P^j = P^{j_1} \cup P^{j_2}$ such that $P^{j_1} \subset P^i$ and the points of P^{j_2} are either dominated by points of P^i (Figure 4.2b) or weakly dominated by interior points in the edges defined by two points of P^i (Figure 4.2c), then $CSUB(u^j)$ dominates $CSUB(u^i)$.*

Dominated CSUBs do not contribute to the computation of the OCSUB. It is therefore useless to determine such CSUBs. However, it is difficult to know that a CSUB is dominated by another one before its computation. In the following, we provide a characterization allowing to know without additional computation that some CSUBs are dominated.

Given a known critical multiplier u, we consider $CSUB(u-\epsilon)$ and $CSUB(u+\epsilon)$ with ϵ small enough such that u is the only critical multiplier associated to a CSUB in the interval $[u-\epsilon, u+\epsilon]$.

Suppose u is 0M and not M1, then there exists at least one solution associated to $CSUB(u-\epsilon)$ which is not $u+\epsilon$-surrogate feasible. Since u is not M1, there does not exist a solution associated to $CSUB(u+\epsilon)$ that is not $u-\epsilon$-surrogate feasible. Thus, by application of Lemma 4, either $CSUB(u+\epsilon)$ dominates $CSUB(u-\epsilon)$, or $CSUB(u+\epsilon) = CSUB(u-\epsilon)$ (case of equivalent solutions). In both cases, the computation of $CSUB(u-\epsilon)$ is useless. We will say that $CSUB(u+\epsilon)$ *weakly dominates* $CSUB(u-\epsilon)$.

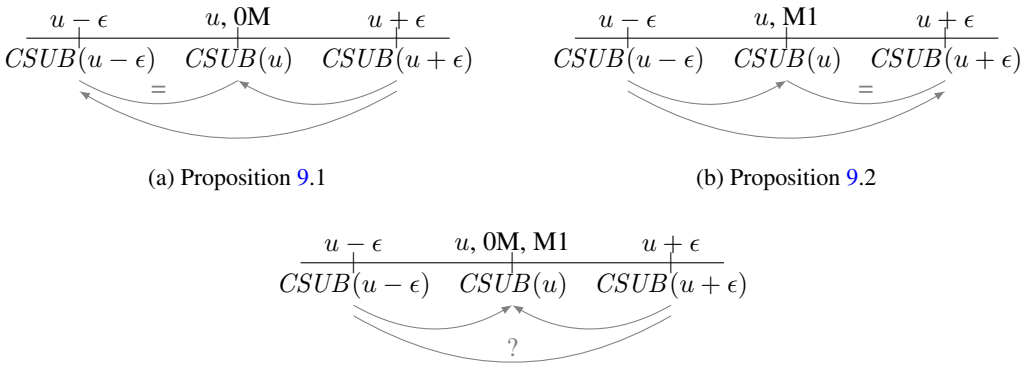
We can do the symmetric analysis in the case u is M1 but not 0M and a similar one if u is both 0M and M1. Proposition 9 summarizes this result.

Proposition 9.

1. *If a multiplier u is 0M and not M1 then $CSUB(u+\epsilon)$ weakly dominates $CSUB(u-\epsilon) = CSUB(u)$.*
2. *If a multiplier u is M1 but not 0M then $CSUB(u-\epsilon)$ weakly dominates $CSUB(u+\epsilon) = CSUB(u)$.*
3. *If a multiplier u is 0M and M1 then $CSUB(u-\epsilon)$ and $CSUB(u+\epsilon)$ may not be comparable. Both weakly dominate $CSUB(u)$.*

If 0 is a critical multiplier, then 0 is 0M and Proposition 9 holds, except that we ignore the comparison with $CSUB(0-\epsilon)$ which is not defined. Similarly if 1 is a critical multiplier, it is M1 and $CSUB(1+\epsilon)$ is not defined.

Proposition 9 is illustrated in Figure 4.4, an arrow from $CSUB(u^j)$ to $CSUB(u^i)$ means that $CSUB(u^j)$ weakly dominates $CSUB(u^i)$. "=" indicates an equality between two CUSBs and "?" indicates that no domination relation between the two CSUBs can be deduce a priori.



(a) Proposition 9.1

(b) Proposition 9.2

(c) Proposition 9.3

Figure 4.4: Properties of 0M and M1 multipliers

Proposition 9 allows only a local analysis around critical multipliers. We will say that we can know *a priori* that a CSUB is weakly dominated. In the following, we apply the result of this proposition on the set of all critical multipliers associated to any CSUB, to obtain a stronger result.

Definition 25. *Let L be a list of critical multipliers sorted by increasing values. We call* 0M-M1 interval *in L any pair $u^0 - u^1$ of successive critical multipliers in $L \cup \{0,1\}$, such that u^0 is 0M and u^1 is M1 and $u^0 < u^1$. 0 and 1 are respectively considered as 0M and M1 multipliers.*

Proposition 10. *We denote by L_{all} the list of all critical multipliers associated to all CSUBs. The only CSUBs that are not weakly dominated* a priori *are $CSUB(u)$ where $u^0 < u < u^1$ for any 0M-M1 interval $u^0 - u^1$ in L_{all}.*

Proof. We consider all possible types for a pair of critical multipliers u^1 and u^2 and we analyze the dominance relation between the CSUBs around these two critical multipliers. We only present the case for which u^1 is 0M (but not M1). The proof is similar if it is M1 or, 0M and M1. As u^1 is 0M, Proposition 9 implies that $CSUB(u^1+\epsilon)$ weakly dominates $CSUB(u^1) = CSUB(u^1-\epsilon)$. As there is no critical multiplier between u^1 and u^2, $CSUB(u^1+\epsilon) = CSUB(u^2-\epsilon)$. The question is next whether $CSUB(u^1+\epsilon)$ is weakly dominated or not by $CSUB(u^2+\epsilon)$ or $CSUB(u^2)$. In all cases, the answer comes from the application Proposition 9 for the multiplier u^2.

1. If u^2 is 0M and not M1, then $CSUB(u^1+\epsilon)$ is weakly dominated by $CSUB(u^2+\epsilon)$.
2. If u^2 is M1 and not 0M, then $CSUB(u^1+\epsilon)$ weakly dominates $CSUB(u^2+\epsilon)$.
3. If u^2 is 0M and M1, then $CSUB(u^1+\epsilon)$ weakly dominates $CSUB(u^2)$.

The only cases for which $CSUB(u^1+\epsilon) = CSUB(u^2-\epsilon)$ is not dominated a priori, happen when u^2 is M1.

Suppose 0 is not a critical multiplier, and we denote the smallest critical multiplier by $u^{\min}$. The same CSUB is therefore obtained for $CSUB(u)$ for $u \in [0, u^{min})$. Depending of the type of

u^{min}, $CSUB(u^{\min} - \epsilon)$ is weakly dominated ($u^{\min}$ is 0M) or not ($u^{\min}$ is M1). The case of the multiplier 1 is symmetric. □

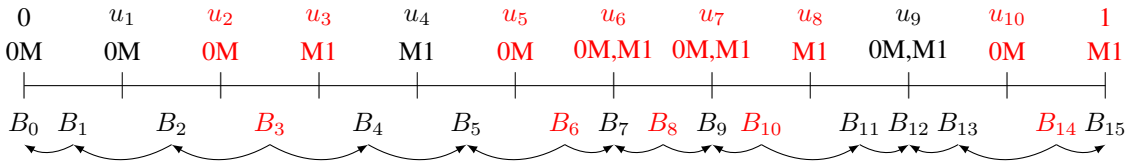


Figure 4.5: Illustration of Proposition 10

Figure 4.5 illustrates Propositions 9 and 10. Dominance between CSUBs is again represented by an arrow, and the CSUBs that are not dominated a priori are denoted in solid red.

4.5.2 0M-M1 interval algorithm

The *0M-M1 interval* algorithm is based on Proposition 10 which states that the only multiplier u such that $CSUB(u)$ is not dominated *a priori* are in the interior of the 0M-M1 intervals in L_{all}. Obviously, the direct use of this result requires to know L_{all}. Knowing only a subset of CSUBs with their associated solutions and critical multipliers, Proposition 7 defines an union of intervals U such that an unknown solution associated to a critical multiplier v is masked on U. To find such an unknown solution x, it is necessary to compute $CSUB(u)$ for $u \in S \setminus U$ where S is the stability interval of x. According to Proposition 7 and Remark 4, U is composed of zero or one interval of type $[0, v^0]$ (or $[v, v^0]$ if v is a M1 multiplier), zero or one interval of type $[v^1, 1]$ (or $[v^1, v]$ if v is a 0M multiplier) and zero or several disjoint intervals of type $[v^1, v^0]$ where the multipliers denoted v^0 and v^1 are known 0M and M1 multipliers. It is interesting to note that the set $S \setminus U$ is an union of 0M-M1 intervals defined by a subset of known multipliers, with the exception of at most one interval for which the extremities are a known critical multiplier and v (unknown).

The 0M-M1 algorithm (Algorithm 5) is based on these observations. An iteration of the algorithm is performed as follows: given a 0M-M1 interval $u^0 - u^1$ in the list of known multipliers, $CSUB(u^0 + \epsilon)$ is computed and the new critical multipliers are stored. We call this operation *the exploration of the 0M-M1 interval*. Since the exploration of a 0M-M1 interval is based on the 0M multiplier, we store the 0M multiplier for each explored 0M-M1 interval, in order to avoid repetitive computations of a same CSUB. Indeed, new multipliers may be found with the computation of a new CSUB. If a new M1 multiplier is inserted between a pair of 0M and M1 multiplier, a new 0M-M1 interval is defined, but its exploration is done by computing the same CSUB.

The initialization is reduced to the computation of $CSUB(0)$. Indeed $CSUB(1)$ is either dominated or redundant with the exploration of a 0M-M1 interval.

4.5.3 Correctness of the *0M-M1 interval algorithm*

In this part we prove the correctness of the *0M-M1 interval algorithm*. In this proof, we denote by L_{alg} the list of known critical multipliers at the end of Algorithm 5. This proof is decomposed in two propositions, its main idea is to show that the list L_{alg} contains at least the minimum number of critical multipliers to guarantee that the OCSUB is obtained as an output of the algorithm. As Algorithm 5 computes $CSUB(u^0 + \epsilon)$ for each 0M-M1 interval $u^0 - u^1$ in L_{alg}, it is necessary for L_{alg} to contain all 0M multiplier that defines a 0M-M1 in the list L_{all} (Proposition 11). Moreover to ensure that $CSUB(u^0 + \epsilon)$ is computed for all u^0 defining a 0M-M1 interval $u^0 - u^1$ in L_{all},

Algorithm 5: 0M-M1 interval algorithm

input : A $2O2DKP$ instance
output: The OCSUB of the instance of $2O2DKP$

```
1 begin
    /* multipliers is sorted in non increasing order, 1 is considered
       as a M1 multiplier                                              */
2   bound ← CSUB(0)
3   multipliers ← compAllCM(CSUB(0)↓) ∪ {1}
4   0MDone ← ∅
5   while there exists a 0M-M1 interval u^0 − u^1 such that u^0 ∉ 0MDone do
6     0MDone ← 0MDone ∪ {u^0}
7     bound ← [(bound − R^2_≧) ∩ (CSUB(u^0 + ε) − R^2_≧)]_N
8     multipliers ← multipliers ∪ compAllCM(CSUB(u^0 + ε)↓)
9   return bound
```

L_{alg} must also contain a M1 multiplier $u^2 > u^0$ such that there does not exist any 0M multiplier in $]u^0, u^2[$ (Proposition 12).

We can note that for a same u^0 defining a 0M-M1 interval $u^0 - u^1$ in L_{all}, several M1 critical multipliers are appropriate to ensure the computation of $CSUB(u^0 + \epsilon)$. To illustrate this, we consider the example of Figure 4.5. The CSUBs to be computed to guarantee that the OCSUB is found, are $CSUB(u + \epsilon)$ with $u = u^2, u^5, u^6, u^7, u^{10}$. L_{alg} must therefore contain these 0M critical multipliers. The M1 critical multiplier that may cause the exploration of the 0M-M1 interval $u^2 - u^3$ (in L_{all}) can be either u^3 or u^4. Indeed there does not exist any 0M critical multiplier in $]u^2, u^3[$ or in $]u^2, u^4[$, thus the explorations of $u^2 - u^3$ and $u^2 - u^4$ are the same: $CSUB(u^2 + \epsilon)$. The other M1 multipliers to find are u^6, u^7 and, u^8 or u^9.

Proposition 11. *Algorithm 5 finds all multiplier u^0 such that there is a 0M-M1 interval $u^0 - u^1$ in L_{all}.*

Proof. Let $u^0 - u^1$ be a 0M-M1 interval in L_{all}, and x^0 be a solution associated to u^0. We show in the following that x^0 (and thus u^0) is found by Algorithm 5. If x^0 is associated to $CSUB(0)$, then x^0 is found with the initialization. If x^0 is not associated to $CSUB(0)$ then it is 0-masked (since x^0 is 0-surrogate feasible). By Proposition 7, the union of intervals on which x^0 is masked is composed of zero or one interval $[0, v^0]$, zero or several intervals of the kind $[v^1, v^0]$, and zero or one interval of the kind $[v^1, u^0]$, where the multipliers denoted v^1 and v^0 are respectively M1 and 0M multipliers in L_{alg}. Since x^0 is 0-masked, it is in particular masked on one interval $[0, v^0]$. Consequently, x^0 is surrogate feasible and not masked by known solutions on a non-empty union of zero or several $]v^0, v^1[$ open intervals, and zero or one $]v^0, u^0]$ interval. The computation of $CSUB(u)$ for u in the interior of one of these intervals, performed in the context of Algorithm 5, necessarily guarantee to obtain x^0. Otherwise new solutions masking x^0 would be found with their associated critical multipliers, and this contradict the assumption that L_{alg} is the list of known critical multipliers at the end of Algorithm 5.

- If the first kind of interval $]v^0, v^1[$ exists, v^0 and v^1 are respectively a 0M and M1 multiplier in L_{alg}. Hence, there is a 0M-M1 interval (in L_{alg}) in $[v^0, v^1]$ and this 0M-M1 interval is explored during the algorithm.
- If the second kind of interval $]v^0, u^0]$ exists, we prove next that there is a 0M-M1 interval in L_{alg} such that the 0M part is in $[v^0, u^0]$.
  - If there is no 0M multiplier bigger than u^0 in L_{alg}, then since 1 is considered as a M1 multiplier, there is a 0M-M1 interval $u'_0 - 1$ in L_{alg} where u'_0 is the biggest 0M multiplier

found.

– If there is at least one 0M multiplier greater than u^0 in L_{alg}, we denote by u^{0SF} the smallest one. Since $u^0 - u^1$ is a 0M-M1 interval in the list L_{all}, then there exists at least one M1 multiplier in $]v^0, u^{0SF}]$ (at least u^1). We prove next that at least one of these multipliers necessarily belong to L_{alg}.
Suppose there is no M1 critical multiplier in $]v^0, u^{0SF}] \cap L_{alg}$, we consider u^2 a M1 multiplier in $]v^0, u^{0SF}] \cap L_{all}$ and x^2 one of its associated solution. If there are several possible critical multipliers u^2, we consider one such that x^2 is not masked by any solutions or pair of solutions associated to any other M1 multipliers of the interval $]v^0, u^{0SF}]$. By Proposition 7, the union of intervals on which x^2 is masked is composed of zero or one interval $[u^2, v'^0]$, zero or several intervals of the kind $[v'^1, v'^0]$, and zero or one interval of the kind $[v'^1, 1]$, where the multipliers denoted v'^1 and v'^0 are respectively M1 and 0M multipliers in L_{alg}. Consequently, x^2 is not masked on a non-empty union of zero or several $]v'^0, v'^1[$ intervals, and zero or one $[u^2, v'^1[$ interval. The computation of $CSUB(u)$ for u in the interior of one of these intervals, performed in the context of Algorithm 5, necessarily guarantee to obtain x^2.
  – If the former kind of interval $]v'^0, v'^1[$ exists, v'^0 and v'^1 are respectively a 0M and M1 multiplier in L_{alg}. Hence, there is a 0M-M1 interval in $[v'^0, v'^1] \cap L_{alg}$ and this 0M-M1 interval is explored during the algorithm. Consequently, x^2 and thus u^2 is found.
  – If the later kind of interval $[u^2, v'^1[$ exists, we have $v^0 < u^0 < u^{0SF} < v'^1$. There exists a 0M-M1 interval in $[v^0, v'^1] \cap L_{alg}$. If its 0M part is smaller than u^0 its exploration is the computation of a CSUB in $]v^0, u^0[$, and consequently x^0 (and u^0) is found. Otherwise, the 0M part is greater than u^0 (and thus greater than u^{0SF}), consequently u^2 and x^2 are found.

□

Proposition 12. *If $u^0 - u^1$ is a 0M-M1 interval in L_{all}, then there is a M1 multiplier $u^2 \in L_{alg}$ such that no 0M critical multiplier exists in $]u^0, u^2[$.*

Proof. Let $u^0 - u^1$ be a 0M-M1 interval in L_{all}. By Proposition 11, $u^0 \in L_{alg}$.

If u^0 is the greatest 0M multiplier in L_{all}, then there is no 0M multiplier in $]u^0, 1[$ and 1 is (considered as) a M1 multiplier in L_{alg}.

If u^0 is not the greatest 0M multiplier in L_{all}, we denote by u^{0S} the smallest 0M multiplier in L_{all} such that $u^0 < u^{0S}$. Since 1 is a M1 critical multiplier, there exists a 0M-M1 interval in $L_{all} \cap [u^{0S}, 1]$. According to Proposition 11 the 0M part of this interval is also found by the algorithm. Thus there is at least one 0M multiplier greater than u^0 in L_{alg}, we denote by u^{0SF} the smallest one. Since there is no 0M multiplier in $L_{alg} \cap]u^0, u^{0SF}[$, there does not exist any M1 critical multiplier in $L_{all} \cap]u^{0S}, u^{0SF}[$ (otherwise according to Proposition 11 a 0M critical multiplier would be found in $]u^0, u^{0SF}[$). Thus the M1 multipliers in $]u^0, u^{0SF}]$ and in $]u^0, u^{0S}]$ are the same.

We show in the following that there is a M1 multiplier in $L_{alg} \cap]u^0, u^{0SF}[$. We consider u^2 a M1 multiplier in $L_{all} \cap]u^0, u^{0SF}]$ and an associated solution x_2 that is not masked by any other solution (or pair of solutions) associated to a M1 multiplier in $]u^0, u^{0SF}]$. By Proposition 7, the union of intervals on which x_2 is masked is an union of zero or one interval $[u^2, v^0]$, zero or several intervals of the kind $[v^1, v^0]$, and zero or one interval of the kind $[v^1, 1]$, where the multipliers denoted v^1 and v^0 are respectively M1 and 0M multipliers in L_{alg}. Consequently, x^2 is not masked on a non-empty union of zero or several $]v^0, v^1[$ intervals, and zero or one $[u^2, v^1[$ interval.

– If the former kind of interval $]v^0, v^1[$ exists: there exists a 0M-M1 interval in $[v^0, v^1] \cap L_{alg}$ and it is thus explored during the algorithm.

– If the latter kind of interval $[u^2, v^1[$ exists: since there is no M1 critical multiplier found in $(u^0, u^{0_{SF}}]$ then $u^{0_{SF}} < v^1$. Thus there exists a 0M-M1 interval in $[u^{0_{SF}}, v^1] \subset [u^2, v^1]$ and it is thus explored during the algorithm.

□

Algorithm 5 is applied on the instance of $2O2DKP$ introduced in Example 5.

Example 8. *At the beginning* multipliers *contains the M1 multiplier 1. The first bound set to be computed is* $CSUB(0) = B_1$. x^1 *and* x^2 *compose this bound set. Then* $\frac{1}{4}$ *and* $\frac{3}{4}$ *(of type 0M) are added to* multipliers.

The only 0M-M1 interval in multipliers *is* $\frac{3}{4} - 1$. *Then the next CSUB computed is* $CSUB(\frac{3}{4} + \epsilon) = B_2$. *This is composed of* x^7 *and* x^8. *Thus the multiplier* $\frac{8}{13}$ *(of type M1) is added to* multipliers *(represented in Figure 4.6).*

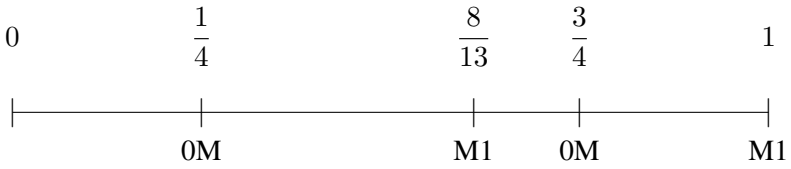


Figure 4.6: Content of *multipliers* after the computation of B_2

There is two 0M-M1 intervals in multipliers *now:* $\frac{1}{4} - \frac{8}{13}$ *and* $\frac{3}{4} - 1$. *Since* $CSUB(\frac{3}{4} + \epsilon)$ *has already been computed then we do not compute it again.* $CSUB(\frac{1}{4} + \epsilon)$ *has not been computed before. So we compute it, it is* B_3 *composed of* x^2 *and* x^5. *The new 0M multiplier* $\frac{7}{16}$ *is added to* multipliers.

$\frac{7}{16} - \frac{8}{13}$ *is the only new 0M-M1 interval. Since* $CSUB(\frac{7}{16} + \epsilon) = B_4$ *has not been computed yet, we compute it now. It is composed of* x^2 *and* x^6. *The new M1 multiplier* $\frac{7}{16}$ *is added to* multipliers. *The content of* multipliers *is represented in Figure 4.7.*

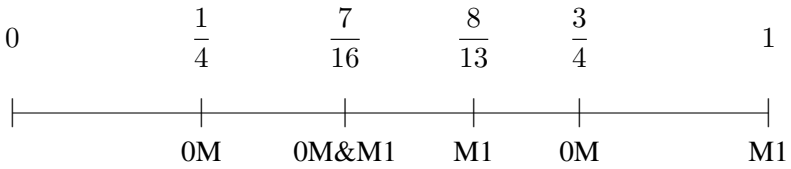


Figure 4.7: Content of *multipliers* after the computation of B_4

The three 0M-M1 intervals found are $\frac{1}{4} - \frac{7}{16}$, $\frac{7}{16} - \frac{8}{13}$ *and* $\frac{3}{4} - 1$. *All of them have already been explored. So the algorithm ends.*

4.5.4 Initialization of the 0M-M1 interval algorithm

The list of critical multipliers of the 0M-M1 interval algorithm can be initialized using the dichotomic algorithm we presented in Section 4.2 (Page 95). We showed that this algorithm does

generally not give the OCSUB and that it may stop prematurely, however it finds surrogate-feasible solutions that defines edges of the OCSUB with their associated critical multipliers. The early knowledge of these critical multipliers permits next to avoid the enumeration of weakly dominated CSUBs. Finally, each step of the dichotomic method consists in solving the dual surrogate problem for $2DKP$, which is less time consuming than to compute a CSUB according to our observations.

The next section presents a heuristic derived from the *0M-M1 interval* algorithm, and a heuristic from the literature.

4.6 Heuristic methods for the OCSUB

4.6.1 A heuristic based on the *0M-M1 interval* algorithm

We describe here a method to compute an approximation of the OCSUB, based on the *0M-M1 interval* algorithm and called *0MM1H*. This heuristic works the same way as the *0M-M1 interval* algorithm, initialized by the dichotomic method, except that the number of iterations is limited to h. Thus all 0M-M1 interval will generally not be explored. Hence we have to define how we choose the 0M-M1 interval to explore at each iteration. If there are several 0M-M1 intervals $u_0 - u_1$ to explore, then we choose the one for which u_1 was part of the most 0M-M1 intervals previously explored. In case of equality we choose the lowest u_0. The idea is to explore improving CSUBs immediately.

If there does not exist any 0M-M1 interval to explore, then the method stops even if the maximum number of iterations is not achieved. Indeed the OCSUB is found, any additional computation of CSUB is therefore useless.

An alternative and more intuitive heuristic, with a lower running time per iteration since it does not analyze the critical multipliers, is now presented.

4.6.2 The *SurrogateFamily* heuristic

The *SurrogateFamily* heuristic (*SF*) has been introduced in the master thesis of Perederieieva (2011) (see Section 2.5.2). *SurrogateFamily* has not been designed with the aim to compute the OCSUB, its purpose is to compute an upper bound set in a short computational time. Thus, there is no guarantee to determine the OCSUB using this method, even with a large number of iterations.

The principle of *SurrogateFamily* is simple. It computes h CSUBs: $\{CSUB(u_1), \ldots, CSUB(u_h)\}$ where the multipliers $\{u_1, \ldots, u_h\}$ are equidistant in the interval $[0, 1]$. Obviously if we increase the value of h, a tighter upper bound set will be computed, but with a larger computational cost.

In *SurrogateFamily*, the choice of the multipliers is completely static. No information is used to avoid the computation of dominated CSUBs. In particular, even if the OCSUB is obtained, *SurrogateFamily* will continue its execution until the last iteration.

4.7 Computation of the OSUB

The OSUB is a non-convex upper bound set which is tighter than the OCSUB. While OCSUB is based on the computation of CSUBs, OSUB is based on SUBs, see Definition 21 (Page 95). Since $SUB(u)$, for $u \in [0, 1]$, is defined by all the nondominated points of $2OSR(u)$ (whereas $CSUB(u)$ is defined only by the extreme supported nondominated points), the computation of $SUB(u)$ is more time consuming than the one of $CSUB(u)$. Therefore, the computation of the

OSUB is also more time consuming than for the OCSUB. In this section we highlight the differences between the computation of the OCSUB and the OSUB.

The notion of critical multipliers and stability interval in Section 4.3.1 are defined for solutions and can thus be indifferently used when computing the OSUB or the OCSUB. The principal difference between the computation of those two upper bound sets comes from the condition under which two CSUBs and two SUBs are different. Lemma 2 (page 98) states a necessary but not sufficient condition for two CSUBs to be different. This lemma can easily be adapted for two SUBs, as follows.

Lemma 5. *Let $u^i \in [0,1]$ and $u^j \in [0,1]$ be two different multipliers, if $SUB(u^i)$ and $SUB(u^j)$ are different then at least one of the efficient solutions of $2OSR(u^i)$ and $2OSR(u^j)$ is feasible for one problem and not for the other one.*

As for Lemma 2, the converse of Lemma 5 is not true because of the potential equivalent solutions. The only difference between Lemmas 2 and 5 is that all nondominated points are considered in the latter while only the extreme supported ones are considered in the former. This difference in the range of considered solutions implies a difference in the characterization of the interval on which a given solution is masked (Proposition 7 at page 100 when computing the OCSUB). When computing the OSUB, the interval on which a solution is masked is defined by Proposition 7, except the points (1)(iii), (2)(iii) and (3)(iii). Indeed those last points are not longer needed since a solution cannot be masked by a pair of solutions (a SUB is not convex). The same way as when computing the OCSUB, one extremity of each interval on which a solution is masked is a critical multiplier when computing the OSUB. Thus the principles behind the *total enumerative* and *0M-M1 interval* algorithms remain valid to compute the OSUB.

The algorithms defined to compute the OCSUB can easily be adapted to the computation of the OSUB. The only change is that the computation of the CSUBs (convex relaxation of $2OSR(u)$, for $u \in [0,1]$) is replaced by the computation of the SUBs (exact solution of $2OSR(u)$, for $u \in [0,1]$). We can also remark that the heuristics method for the computation of the OCSUB can be adapted similarly for the computation of the OSUB.

The next section present experimental results on the algorithms and heuristics approaches to compute the OCSUB and the OSUB.

4.8 Numerical experiments

4.8.1 Protocol

The numerical experiments aim (1) to show experimentally the performances of the proposed algorithms, (2) to observe if an algorithm presents in general better performances than its competitor, (3) to examine the behavior of the algorithm when the number of variables increases, and (4) to note, if it exists, the influence of correlations between the values of the instances on the behavior of the algorithm. To discuss about those factors, six descriptors are elaborated. The two first concern the exact algorithms (*TotalEnumerative* and *0M-M1 interval*),

Descriptor 1. The number of computed CSUBs. A low number shows a fast convergence to the OCSUB.

Descriptor 2. CPU time to get the OCSUB.

The two following are for the approximation algorithms (*0MM1H* and *SurrogateFamily*).

Descriptor 3. Quality of the approximation. The quality assessment of an approximated bound set will be stated in comparison with the OCSUB. Here, an area measure (called $\mathcal{A}$-metric) is proposed and reported in terms of percentage. A small $\mathcal{A}$-metric value indicates an approximation close to the OCSUB.

Descriptor 4. CPU time to get the approximated OCSUB.

The two last are for the comparison of OSUB and OCSUB (*0MM1H* and *SurrogateFamily*).

Descriptor 5. Quality of the OCSUB compared to the OSUB. Here, an area measure (called $\mathcal{A}'$-metric) is proposed and reported in terms of percentage. A small $\mathcal{A}'$-metric value indicates an approximation close to the OCSUB.

Descriptor 6. CPU time to get the OCSUB and the OSUB.

All our implementations require to fix a value for ϵ (small positive number used in Algorithms 3, 4 and 5). The smallest value manageable by our solver's implementation is $\varepsilon = 10^{-9}$ (the recommended value returned using the computing method described in Appendices B.1 is smaller). Next, the methods approximating the OCSUB require to fix a value for the parameter h. The following values have been used: 10, 50, 100.

The computer used for the experiments is equipped with a Intel Core i7 2640M 2.8 Ghz processor with 8 GB of RAM, and runs under Mac OS X Lion 10.7.5. All algorithms have been implemented in C++. The binaries have been obtained using the compiler GCC version 4.2.1. with the optimizer option -O3. *Combo* from (Martello et al., 1999) has been used to solve single-objective uni-dimensional knapsack problems and the algorithm from (Jorge, 2010) has been used to solve the $2OKP$ in the computation of SUBs.

4.8.2 Numerical instances

All algorithms have been evaluated on a dataset composed of 28 instances available on the MCDMLib [1], and organized in three groups:

Group 1: 6 instances of various sizes, randomly generated with the same generator, without any specific correlation. `ZTL100, ZTL250, ZTL500 and ZTL750` are 4 instances picked from the collection maintained by Zitzler and Laumanns [2]. `ZTL28 and ZTL105` are 2 additional instances derived respectively from ZTL100 and ZTL250, where $\omega_i \approx 0.5 \sum_{j=1}^{n} w_{ij}$, $i = 1, \ldots, m$. The number following ZTL in the name gives the number of variables.

Group 2: 15 correlated instances with 28 variables, introduced by Perederieieva (2011). `A1, A2, A3, A4, D1, D2, D3, D4, kp28W-ZTL, kp28, kp28-2, kp28W, kp28W-Perm, kp28c1W-c2ZTL` and `kp28cW-WZTL` are extention of $2DKP$ instances available on the OR-library [3] where a second objective has been added with respect to several definitions of correlation to obtain a $2O2DKP$ instance. The second objective of `kp28W-Perm` is a permutation of the coefficients of the first objective function. The second objective function is not correlated to the first objective function, neither the constraints for the instances `kp28W-ZTL, kp28, kp28-2, kp28W, kp28c1W-c2ZTL`. For the other instances, the second objective is correlated with the first one. Further details are given in (Perederieieva, 2011).

1. http://www.mcdmsociety.org/MCDMlib.html
2. http://www.tik.ee.ethz.ch/sop/download/supplementary/testProblemSuite/
3. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html

Group 3: 7 additional correlated instances with 105 variables, obtained following the rules described for the Group 2. `W7BI-rnd1-1800`, `W7BI-rnd1-3000`, `W7BI-tube1-1800`, `W7BI-tube1-3000`, `W7BI-tube1-asyn`, `W7BI-tube2-1800`, `Wcollage-tube` are summarized in Table 4.3.

instance	(a)	(b)	(c)	(d)
W7BI-rnd1-1800	Weing7	Osorio's method	0.48	0.50
W7BI-rnd1-3000	Weing7	Osorio's method	0.80	0.84
W7BI-tube1-1800	Weing7	anti-correlated	0.50	0.50
W7BI-tube1-3000	Weing7	anti-correlated	0.80	0.84
W7BI-tube1-asyn	Weing7	anti-correlated	0.80	0.20
W7BI-tube2-1800	Weing7	anti-correlated	0.48	0.50
Wcollage-tube	Weing1	anti-correlated	0.50	0.50

Table 4.3: Information about instances. Column (a) gives the name of the original single-objective $2DKP$. Column (b) indicates the way to generate the second objective: negatively correlated with the first one or, done according to the method reported in (Osorio and Cuaya, 2005). Column (c) reports the ratio $\frac{\omega_1}{\sum_{j=1}^{n} w_{1j}}$ for the first dimension (the first constraint). The column (d) is similar to the column (c), for the second dimension.

4.8.3 Comparison between *TotalEnumerative* and *0M-M1 interval*

In this section, we compare the two methods that determines the OCSUB. In order to measure the impact of its initialization, the *0M-M1 interval* algorithm is tested with and without its initialization. All results are summarized in Figure 4.8. In this figure, we denote by respectively *TotEnum*, *0MM1* and *0MM1 + init*, the *TotalEnumerative* algorithm, the *0M-M1 interval* without initialization, and the *0M-M1 interval* algorithm using the initialization.

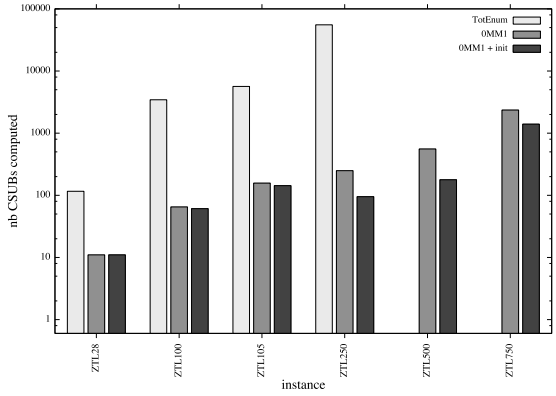
In Figure 4.8, the y-axis is reported on a logarithmic scale for all graphics. The left column reports the numbers of computed CSUBs (descriptor 1), and the computational times (descriptor 2) are on the right column. The line on the top, the middle and the bottom correspond respectively to instances of Group 1, 2 and 3.

For all three groups of instances, the two versions of *0M-M1 interval* are always faster than *TotalEnumerative*. The profile of the curves of time are similar. The difference of execution time increases with the size of the instance. This especially allows *0M-M1 interval* without initialization to determine the OCSUB for ZTL500 and ZTL750 instances with less than 700 seconds of CPU time, while *TotalEnumerative* is unable to compute it in less than one hour.

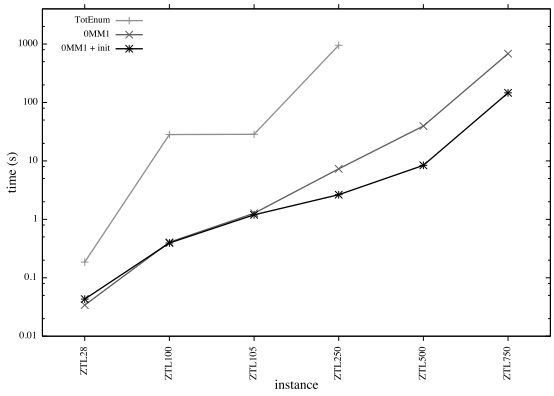
The additional cost of the initialization in the *0M-M1 interval* algorithm is largely balanced for most instances. The only exceptions concerns small instances which solution time is particularly low with or without initialization (less than 0.1s). Moreover, the benefit of the initialization increases significantly with instance size, as can be seen in Figure 4.8 (b).

For groups 2 and 3, an obvious correlation appears between the required computational time and the number of CSUBs to compute. On the other hand, there is no particular difficulty for the algorithms to deal with numerical instances presenting various correlations.

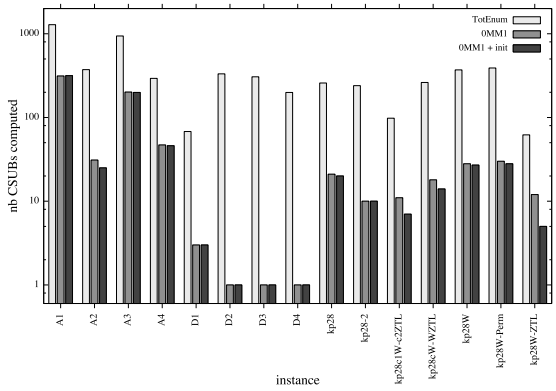
For four particular instances (`D2`, `D3`, `D4` and `tube-asyn`), *TotalEnumerative* algorithm computes a large number of CSUBs, while *0M-M1 interval* only computes 1. This difference is of course reverberated on the computational time.



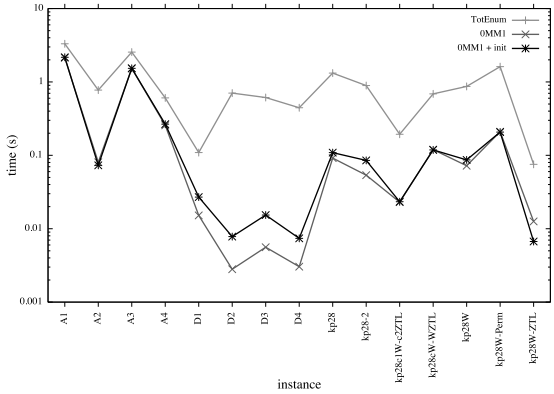
(a) Comparison for the number of computed CSUBs for the instances of Group 1



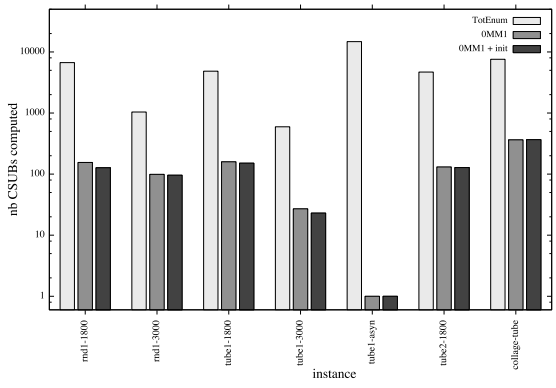
(b) Comparison of computational times for the instances of Group 1



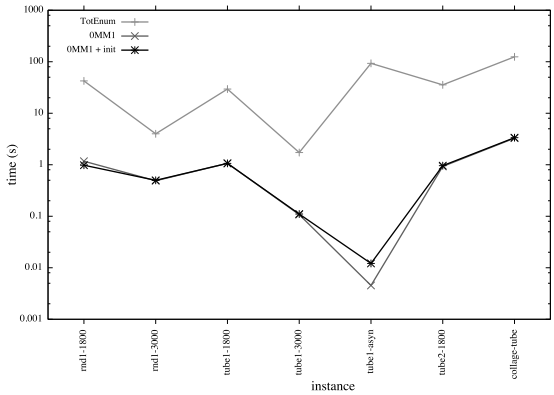
(c) Comparison for the number of computed CSUBs for the instances of Group 2



(d) Comparison of computational times for the instances of Group 2



(e) Comparison for the number of computed CSUBs for the instances of Group 3



(f) Comparison of computational times for the instances of Group 3

Figure 4.8: Comparison of the *TotalEnumerative* and *0M-M1 interval* algorithms.

In conclusion of the analysis, *0M-M1 interval* outranks *TotalEnumerative*, and shows no particular difficulty to deal with instances with correlations. The initialization procedure allows to largely reduce the computational time for large instances. The recommended algorithm for com-

puting the OCSUB is thus the 0M-M1 interval algorithm with initialization. However, the computational time of the OCSUB remains far to be negligible, in particular for big instances. A better trade-off between computational time and quality of the obtained upper bound set should be used, to be embedded as a component of an implicit enumeration algorithm aiming to generate a complete set of efficient solutions for a $2O2DKP$.

4.8.4 $\mathcal{A}$-metric as a quality measure

In order to evaluate the quality of the heuristics, we define a quality indicator. It aims to measure the gap between a heuristic upper bound set and the OCSUB. According to Definition 12, an upper bound set B for Y_N should verify $Y_N \subset (B - \mathbb{R}^2_{\leqq})$. If we consider an upper bound set H obtained by *0MM1H* or *SurrogateFamily* heuristics, we necessarily have $(OCSUB - \mathbb{R}^2_{\leqq}) \subset (H - \mathbb{R}^2_{\leqq})$. We denote next by $\mathcal{A}(V)$ the area of the polyhedra $V \cap \mathbb{R}^2_{\leqq}$. A possible way to measure the gap between H and the OCSUB could be to compare $\mathcal{A}(\text{H}-\mathbb{R}^2_{\leqq})$ and $\mathcal{A}(OCSUB - \mathbb{R}^2_{\leqq})$.

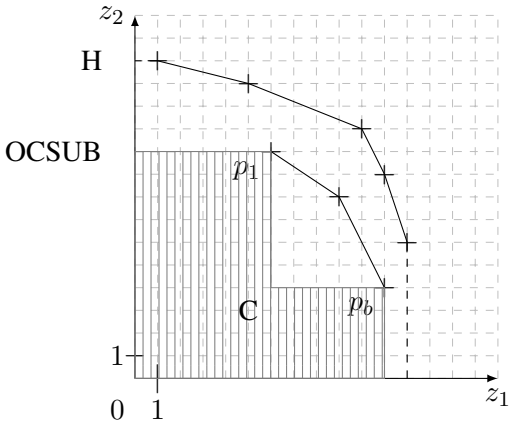


Figure 4.9: Area measure for the comparison

In order to have the same scale for all instances, we could consider the following ratio.

$$\frac{\mathcal{A}(\text{H} - \mathbb{R}^2_{\leqq}) - \mathcal{A}(OCSUB - \mathbb{R}^2_{\leqq})}{\mathcal{A}(OCSUB - \mathbb{R}^2_{\leqq})}$$

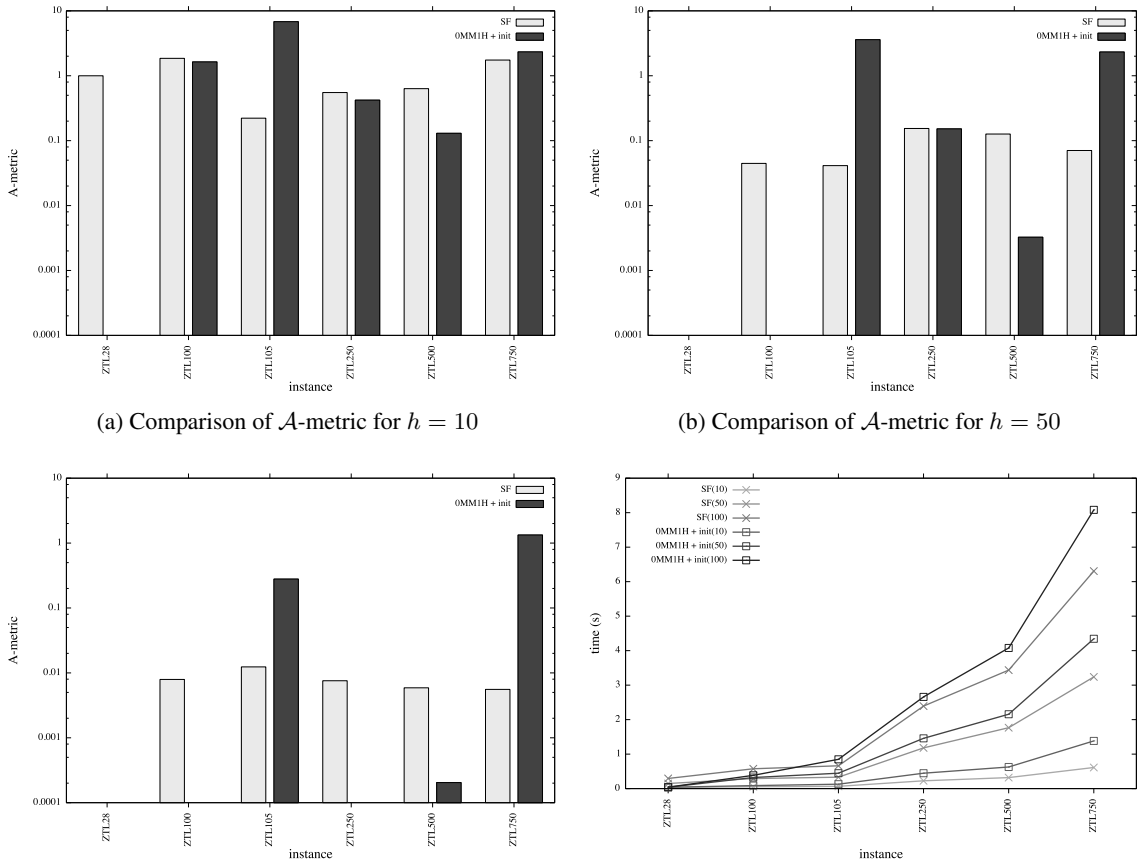
However, such an indicator would be very sensitive to the range of data. In particular, for instances for which the objective values of nondominated points are very large, there would be a large intersection between $\mathcal{A}(\text{H}-\mathbb{R}^2_{\leqq})$ and $\mathcal{A}(OCSUB - \mathbb{R}^2_{\leqq})$, for all possible upper bound sets H. Consequently, an indicator defined this way would necessarily return small values.

In order to highlight the differences between bound sets, we propose to define our quality indicator by putting aside common parts between $\mathcal{A}(\text{H}-\mathbb{R}^2_{\leqq})$ and $\mathcal{A}(OCSUB - \mathbb{R}^2_{\leqq})$, for all possible upper bound sets H. Figure 4.9 illustrates the definition of this indicator. The OCSUB is composed of b extreme points $\{p^1, \ldots, p^b\}$ lexicographically ordered. The search area at the bottom left of p^1 is common to all possible upper bound set H and to the OCSUB, thus we do not need to consider it in the measure. The same statement holds for p^b. Thus $C := (p^1 - \mathbb{R}^p_{\leqq}) \cup (p^b - \mathbb{R}^p_{\leqq})$ is included into $(H - \mathbb{R}^2_{\leqq})$ for all possible upper bound set H. In the following, we call *$\mathcal{A}$-metric* (Figures 4.10 to 4.12) the following measure of quality of a heuristic bound set H:

$$\frac{\mathcal{A}(H - \mathbb{R}^p_{\leqq}) - \mathcal{A}(\text{OCSUB} - \mathbb{R}^p_{\leqq})}{\mathcal{A}(\text{OCSUB} - \mathbb{R}^p_{\leqq}) - \mathcal{A}(C)} * 100$$

This quality indicator has the effect to highlight with large values the difference between bound sets.

4.8.5 Comparison between *0MM1H* and *SurrogateFamily* heuristic



(a) Comparison of $\mathcal{A}$-metric for $h = 10$

(b) Comparison of $\mathcal{A}$-metric for $h = 50$

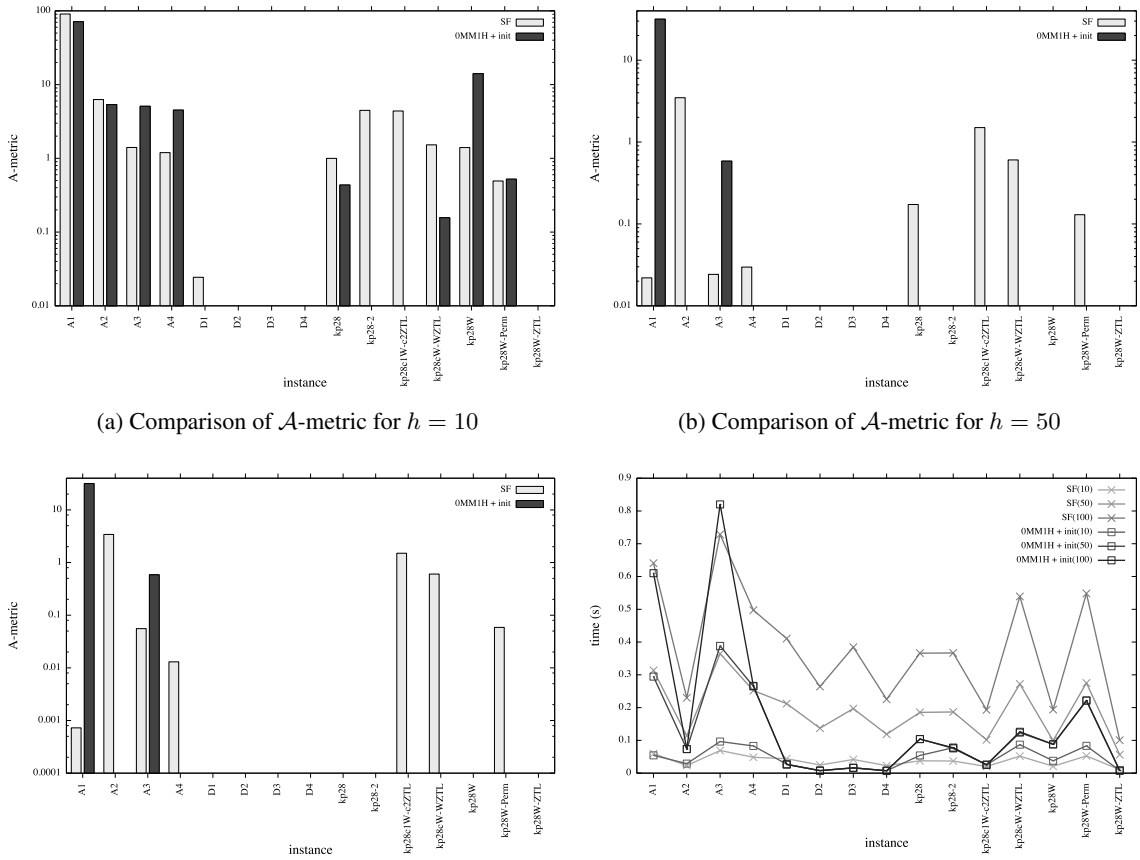
(c) Comparison of $\mathcal{A}$-metric for $h = 100$

(d) Comparison of computational times

Figure 4.10: Comparison of both heuristics *0MM1H + init* and *SurrogateFamily*, for $h = \{10, 50, 100\}$, regarding the $\mathcal{A}$-metric and the computational time, for the Group 1.

The results are summarized in Figure 4.10, 4.11 and 4.12 respectively for instances of groups 1, 2, and 3. In each figure, the graphics (a), (b) and (c) compare both heuristics according to descriptor 3 for respectively $h = 10$, $h = 50$ and $h = 100$. The y-axis is logarithmic. For an instance and a heuristic, if a bar is not visible, it means that the obtained upper bound set is the OCSUB or so close to it that the difference is insignificant. The graphics (d) present the computational times in six curves (descriptor 4), the names of the curves are composed by the name of the heuristic and the parameter h. To read this graphics, it is important to notice that, for a same heuristic, the curve for a given value of h cannot be lower than the one for a smaller value of h (h is the maximum number of iterations allowed). Thus in graphics (d), for a same heuristic, the curve at the bottom corresponds to $h = 10$ and the one at the top to $h = 100$.

For all groups of instances, both algorithms behave the same way in terms of computational time. Generally the computational time of *0MM1H* is slightly larger than for *SurrogateFamily*, because of the initialization and the analysis of the critical multipliers. For some instances, es-



(a) Comparison of $\mathcal{A}$-metric for $h = 10$

(b) Comparison of $\mathcal{A}$-metric for $h = 50$

(c) Comparison of $\mathcal{A}$-metric for $h = 100$

(d) Comparison of computational times

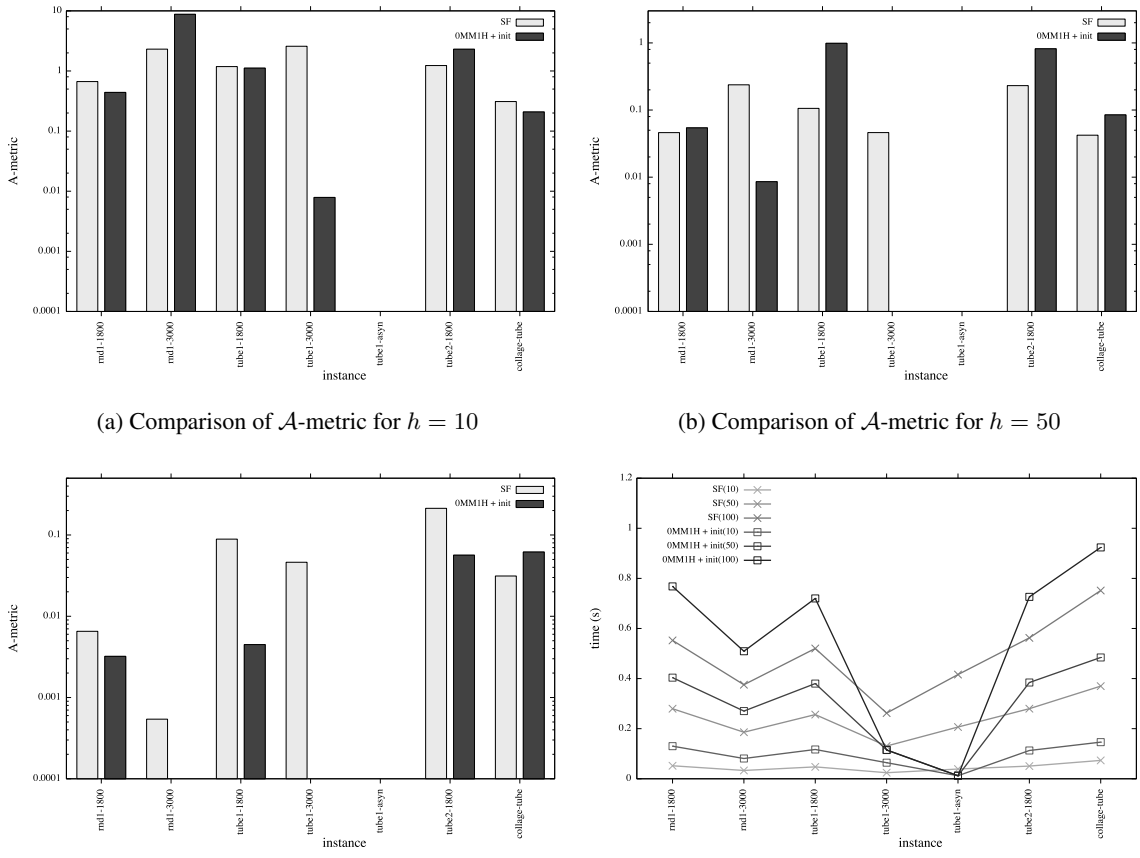
Figure 4.11: Comparison of both heuristics *0MM1H + init* and *SurrogateFamily*, for $h = \{10, 50, 100\}$, regarding the $\mathcal{A}$-metric and the computational time, for the Group 2.

sentially from Group 2, *0MM1H* appears faster. This is due to the early obtention of the OCSUB by *0MM1H*. Consequently, this heuristic stops its execution, while *SurrogateFamily* continue its execution until the final iteration.

In 85% of the cases *0MM1H* performs better than *SurrogateFamily*, according to descriptor 3. When *0MM1H* is performed with $h = 100$, the OCSUB is obtained for 16 of the 28 instances, doing less iterations than h.

The quality of the upper bound set found by *0MM1H* does not seem to be related to the size of the instances. Indeed in Group 1, the performance of *0MM1H* is outranked by *SurrogateFamily* for the instances of size 105 and 750 variables, however the opposite occurs for the instances of size 250 and 500. For the instances of Group 2, *0MM1H* gives upper bound set of good quality in a smaller number of iterations than *SurrogateFamily*, except for two instances (`A1` and `A3`). The difference between both heuristics is also important for several other instances (for example `A2`), in favor of *0MM1H*. In Group 3, both heuristics are almost equivalent in terms of descriptor 4. Finally, the best heuristic for an instance is not necessarily the same for every value of h. The instance `tube1-1800` is an example of this phenomenon: for $h = 10$ both heuristics are equivalent, for $h = 50$ *SurrogateFamily* performs better and the opposite occurs for $h = 100$.

The difference of behavior can be explained by the the use of uniformly distributed multipliers in *SurrogateFamily*. Therefore, the obtained bound set is constructed from various CSUBs. In



(a) Comparison of $\mathcal{A}$-metric for $h = 10$

(b) Comparison of $\mathcal{A}$-metric for $h = 50$

(c) Comparison of $\mathcal{A}$-metric for $h = 100$

(d) Comparison of computational times

Figure 4.12: Comparison of both heuristics *0MM1H + init* and *SurrogateFamily*, for $h = \{10, 50, 100\}$, regarding the $\mathcal{A}$-metric and the computational time, for the Group 3.

0MM1H, the values of the multipliers are determined with the purpose to improve locally the bound set. Thus, with a low number of computed CSUBs, we obtain a bound set that may be very tight locally but not globally. This explanation is confirmed by the results collected for Group 2. In conclusion of the analysis, *0MM1H* is competitive for every size of instances and every tested h, except for some rare instances. A deep analysis of those instances may allow us to find a better order for the exploration of the 0M-M1 intervals. The heuristic using this order could provide better results.

4.8.6 Comparison between the OCSUB and the OSUB

Figure 4.13 shows the computational time required to compute the OCSUB and the OSUB, using the *0M-M1 interval* algorithm with the initialization, for the instances of the three groups. The top, middle and bottom rows present the results obtained for, respectively the Group 1, the Group 2 and the Group 3 of instances. The left column shows the computational time spent on the computation of the two upper bound sets. The y-scale of those figures is logarithmic. The second column presents a measure similar to the $\mathcal{A}$-metric, called the $\mathcal{A}$'-metric. This metric indicates the

gap between the OSUB and the OCSUB. It is calculated by the formula:

$$\frac{\mathcal{A}(\text{OCSUB} - \mathbb{R}^p_{\geqq}) - \mathcal{A}(\text{OSUB} - \mathbb{R}^p_{\geqq})}{\mathcal{A}(\text{OSUB} - \mathbb{R}^p_{\geqq}) - \mathcal{A}(C')} * 100$$

where $C := (p'^1 - \mathbb{R}^p_{\geqq}) \cup (p'^b - \mathbb{R}^p_{\geqq})$ with p'^1 and p'^b the lexicographic extreme points of OSUB.

The high values of the $\mathcal{A}$'-metric in Figures 4.13b, 4.13d and 4.13f show that the OSUB is an upper bound set significantly tighter than the OCSUB. This gap between those two upper bound set does not seem to depend on the size of the instances, neither on the correlation of those instances. One instance can be particularly remarked, the `kp28W-ZTL`, for which the value of the $\mathcal{A}$'-metric is particularly high. Indeed the OSUB of this instance is composed of only two points, the value of the $\mathcal{A}$'-metric is then not defined (division by zero since $\mathcal{A}(\text{OSUB} - \mathbb{R}^p_{\geqq}) = \mathcal{A}(C')$). The OCSUB being a convex upper bound set, the gap between the OCSUB and the OSUB is infinitely larger than the area $\mathcal{A}(\text{OSUB} - \mathbb{R}^p_{\geqq}) - \mathcal{A}(C')$.

The computational time required to compute the OSUB is in average 96 longer than the one required to compute the OCSUB (see Figures 4.13a, 4.13c and 4.13e). Indeed, the OSUB computes SUBs which are more time consuming to compute than the CSUBs for the OCSUB. Moreover, the OSUB requires the solution of a number of surrogate relaxation which is more than twice the one required for the OCSUB (2.2 times in average and up to 11.3). The difference of computational time seems to increase exponentially with the size of the instances (see Figure 4.13a). The variations of computational time between the instances are the same for the computation of the OCSUB as for the computation of the OSUB.

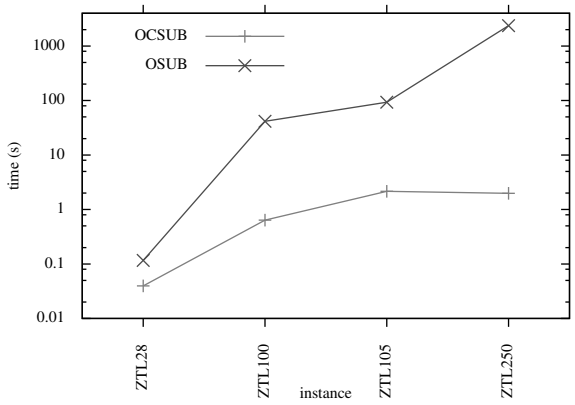
Even if the quality of the OSUB is significantly tighter than the OCSUB, the computational time required to compute the OSUB is considerably higher. Because of this computational cost, it is not reasonable to consider the OSUB as upper bound set in an implicit enumerative method.

Obviously the OSUB can be approximated by the heuristic methods introduced in Section 4.6. For a sake of readability of this manuscript, the experimental results of the heuristic methods approximating the OSUB are presented in Appendix B.3. The behavior of those methods to approximate the OSUB are very similar to the one obtained when approximating the OCSUB. As for the exact computation, the computational time required by those heuristic methods is significantly higher when approximating the OSUB than the OCSUB.

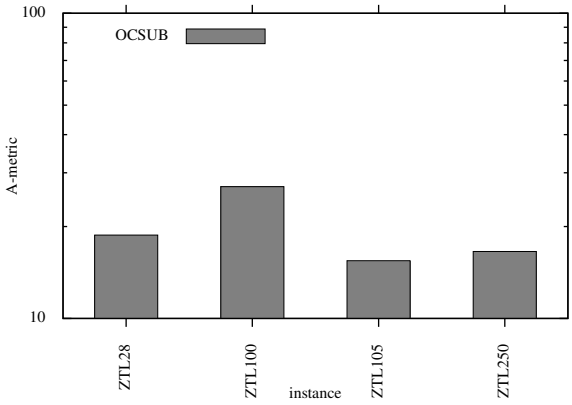
4.9 Conclusion

The optimal convex surrogate upper bound (OCSUB) is the tightest bound set that it is possible to define and to compute with the convex surrogate relaxation. This chapter gives a formal definition of the OCSUB, introduces two exact algorithms for computing the OCSUB, and demonstrates their correctness. A heuristic version is also introduced for a use when the computational time is limited. The numerical experiments clearly demonstrate the effectiveness of the algorithm called *0M-M1 interval*, using the initialization procedure, for the computation of the OCSUB, and show that the algorithm is not sensitive to the presence of correlation in the numerical instances.

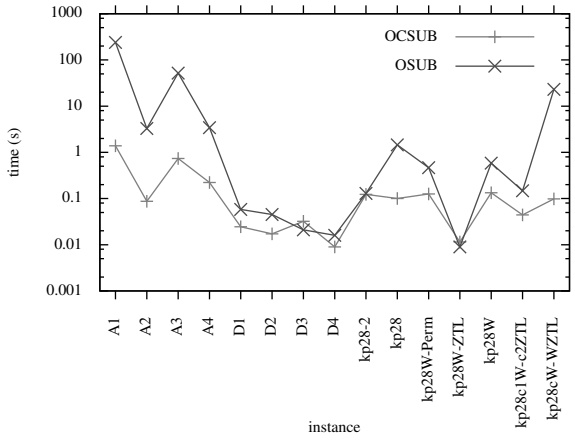
The OCSUB finds its usage naturally in the context of an implicit enumerative method for solving a bi-objective bi-dimensional binary knapsack problems ($2O2DKP$). For example, OCSUB is ready to be embedded in an evaluation component aiming to prune or not a node of a branch-and-bound algorithm, or a state in dynamic programming algorithm. However, it is not surprising to note that the computation of the OCSUB may require a significant CPU time, even



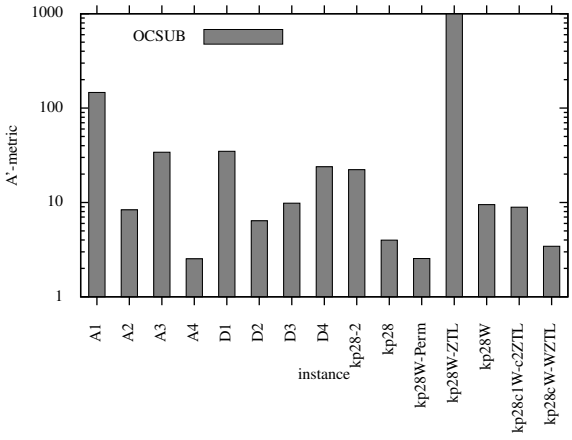
(a) Comparison of computational times for the instances of Group 1



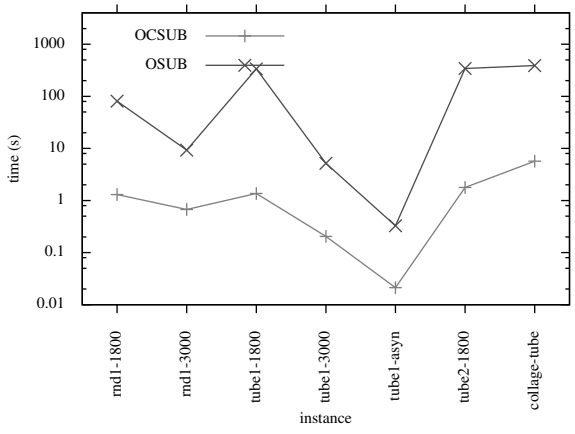
(b) Gap between the OCSUB and the OSUB for the instances of Group 1



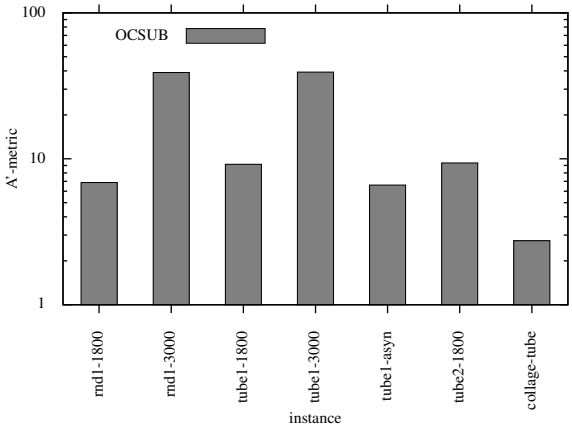
(c) Comparison of computational times for the instances of Group 2



(d) Gap between the OCSUB and the OSUB for the instances of Group 2



(e) Comparison of computational times for the instances of Group 3



(f) Gap between the OCSUB and the OSUB for the instances of Group 3

Figure 4.13: Comparison of the OCSUB and the OSUB, computed using the *0M-M1 interval* algorithm with the initialization.

for mid-size numerical instances. Due to this fact, and in the state of knowledge, it is currently not reasonable to consider such an usage of the OCSUB. The introduction of a heuristic variant for the computation of the OCSUB is therefore justified. The quality of the resulting measured approximation, even for a short CPU time, shows the practical interest of the heuristic algorithm to compute an approximation of the OCSUB. Thus, it appears as a powerful component to encapsulate in an implicit enumeration method, in particular to solve instances of $2O2DKP$. *0MM1H* seems to be a good choice to determine an approximation of the OCSUB, even if its efficiency drops for some rare instances. However, there remain several possible improvements that may change favorably this situation.

The optimal surrogate upper bound set (OSUB) is the tightest (non-convex) upper bound set based on the surrogate relaxation. In this chapter, we have presented the adaptation of the algorithms developed to compute the OCSUB, to compute the OSUB. The adaptation of the *0M-M1 interval* algorithm have been tested experimentally. Since at each iteration of this algorithm, all nondominated points of the surrogate relaxation are searched, the computational time is considerably more important than for the computation of the OCSUB. As a consequence, the use of the OSUB as upper bound set in an implicit enumerative method is not worth considering.

The following prospect allow to expect interesting improvements in terms of the OCSUB computation time, and on the use of the *0MM1H* in an implicit enumeration method. It is linked to the use of these algorithms in an implicit enumeration method, following a re-optimization principle, e.g. between two levels of the enumeration tree. The idea aims to avoid to repeat computations between two successive OCSUBs (or approximations), while the two nodes in the tree vary only by a small subset of variables changing from a free status to fixed. This idea is currently under research.

5

A branch-and-cut method for the bi-objective bi-dimensional knapsack problem

Branch-and-bound methods are more and more used to solve bi-objective combinatorial optimization problems. The upper bound sets for $\bar{Y}_N$ (the nondominated set of the subproblems) are generally based on a relaxation of the problem. Solution methods dedicated to multi-objective knapsack problems generally use either the convex or the LP relaxation. The surrogate relaxation can also be used.

The convex relaxation leads to the tightest convex upper bound set. Moreover its extreme supported efficient solutions are feasible. It is "easy" to solve if the single objective version of the problem can be solved in polynomial or pseudo polynomial time. However the resolution of this relaxation can be time consuming if the single-objective version is not tractable in a reasonable time.

The LP relaxation leads to an upper bound set which is less tight than the convex relaxation. However, its computation is less time consuming, since it can be performed by a parametric simplex method.

The surrogate relaxation is used for single-objective knapsack problems and recently for bi-objective ones. In Chapter 4, we presented two tight upper bound sets for the $2O2DKP$ based on the surrogate relaxation.

In this chapter we aim to design a solution method to solve $2O2DKP$. After the description of the benchmark instances in Section 5.1, Section 5.2 presents a branch-and-bound method embedded in a two phase method. Several relaxations are used and their practical performances are evaluated. The impact of the initialization of the set of feasible solutions is also assessed. Section 5.3 introduces cover inequalities, which are adapted for the multi-objective context in Section 5.4. This latter section also presents the branch-and-cut method and different variants of it. The variants consider different generation mechanism for the cover inequalities. Finally, in Section 5.5, the performances of those variants are assessed.

5.1 Benchmark

All along this chapter, different implementations of the branch-and-bound based and branch-and-cut based methods are assessed on benchmarks of instances. Most of the instances used for our experiments are the same than the ones used in Chapter 4.

In order to have a sample of sizes for the instances, we introduce four new instances: `ZTL150-f250`, `ZTL150-f500`, `ZTL200-f250` and `ZTL200-f500`. The two first ones are composed of 150 variables and the two last ones of 200 variables. These four instances are derived from $2O2DKP$ instances, by randomly selecting items (weights and objective coefficients) and fixing the tightness ratio to 0.5 on each constraint. `ZTL150-f250` and `ZTL200-f250` are derived from `ZTL250` and `ZTL150-f500` and `ZTL200-f500` are derived from `ZTL500`.

The computer used for the experiments is the same as described in Section 4.8.1: equipped with a Intel Core i7 2640M 2.8 Ghz processor with 8 GB of RAM, and runs under Mac OS X Lion 10.7.5. All algorithms have been implemented in C++.

5.2 A branch-and-bound method for the bi-objective bi-dimensional knapsack problem

In this section, we aim to design a branch-and-bound method embedded in a two phase method to solve $2O2DKP$ and test different variants of this method. Firstly, we describe the lower bound set, the preprocessing treatments, the generation of new feasible solutions and the separation strategy, for which the implementation is the same in all variants. Next, we propose different implementations for the upper bound set, the branching strategy and the initialization procedure, respectively in Sections 5.2.2, 5.2.3 and 5.2.4.

5.2.1 Solution method

The method used in this section is a two phase method whose second phase is a branch-and-bound method. The different components of the branch-and-bound method are described below.

The lower bound set

The lower bound set for Y_N used here is the set of potentially nondominated points found so far. Since we are not interested in finding equivalent efficient solutions, the lower bound set is shifted (by the vector (1,1)).

Preprocessing treatments

In order to reduce the number of variables, two preprocessing treatments are applied.

The first one is the global preprocessing treatment presented in (Jorge, 2010) (see Section 2.4.2 for further details). Even if this preprocessing treatment was introduced for $pOKP$, it is easily adapted to $pOmDKP$ by generalizing the notion of dominance between variables. The vector $v_j = (c_j^1, \ldots, c_j^p, -w_{1j}, \ldots, -w_{mj})$ is defined for each item $j \in \{1, \ldots, n\}$. The generalization of the definitions of the dominance relation and of the preferred and dominated sets directly result from the vectors v_j. Moreover, their properties remain unchanged.

The second preprocessing treatment is the one introduced in (Ingargiola and Korsh, 1973) for KP and adapted to $2OKP$ by Visée et al. (1998) and Delort and Spanjaard (2010) (see Section 2.4.2 for further details). The generalization of this preprocessing treatment from $2OKP$ to

$2O2DKP$ is straightforward by considering an appropriate upper bound set. The upper bound set used in this preprocessing treatment, in our method, is the one used during the branch-and-bound execution. This preprocessing treatment is done for each investigated triangle, defined by two adjacent supported nondominated point (see Section 1.4.6), before executing the branch-and-bound method.

Generation of new feasible solutions

An important part of a branch-and-bound method is the ability to find new good solutions. Indeed those solutions may be used to improve the lower bound set and may allow to prune earlier branches of the search-tree. We compare the practical efficiency of three relaxations (convex, LP and surrogate). Generating new feasible solutions is more or less straightforward, depending on the considered relaxation.

The extreme supported efficient solutions of the convex relaxation are feasible for the original problem. Thus when using this relaxation to compute the upper bound set, we obtain feasible solutions of good quality at each node.

As explained in Chapter 4, the solutions of the surrogate relaxation of $2O2DKP$ respects one or the two constraints. Thus when computing a CSUB, the solutions found may respect the two constraints, i.e. be feasible for $2O2DKP$. By performing a simple feasibility check on the solutions associated to CSUB, we obtain feasible solutions for $2O2DKP$.

However when the LP relaxation is used, the extreme supported efficient solutions of this relaxation (defining the upper bound set) are mostly fractional, thus they are very rarely feasible for $2O2DKP$. In our method, a procedure is launched after every computation of the upper bound set to "correct" the extreme supported efficient solutions of the LP relaxation into integer solutions. Let us consider x one of those solutions with a fractional value for at least one variable. If all the fractional variables of x are fixed to 1 (instead of their fractional value), then obviously the capacity constraints would be violated. The quality of the corrected solution depends on the choice of variables to fix to 0 and 1, among the fractional variables. In this method we fix the fractional variables according to the utility $u_j = \dfrac{c_j^1 + c_j^2}{w_{1j} + w_{2j}}$ and we want to favor the variables with a higher u_j. The procedure, applied here, consists in fixing initially all fractional variables to 0. Then for each item $j \in \{1, \ldots, n\}$, in the decreasing order of u_j, the value 1 is affected if the item can be added without exceeding the capacity. Otherwise, the value of this variable is kept to 0. The procedure does not stop as soon as an item does not fit in. Indeed items with a lower u_j might be added in the knapsack without exceeding the capacity.

Separation strategy

The branch-and-bound method explores the nodes in a depth-first search approach. The branch in which the variable is fixed to 1 is explored first. By exploiting the dominance relations presented in (Jorge, 2010), when fixing a variable during the separation procedure, other variables can be fixed without altering the efficient solutions (Jorge, 2010). When fixing a variable to 1, all variables in its preferred set can also be fixed to 1. Symmetrically, when fixing a variable to 0, all the variables in its dominated set can also be fixed to 0.

5.2.2 Comparison of the upper bound sets

The implemented upper bound sets are based on three different relaxations: the convex relaxation, the surrogate relaxation and the LP relaxation. The same relaxation is considered for every

node in the search-tree.

The convex relaxation of $2O2DKP$ is solved using the classical dichotomic search. For each chosen direction, the resulting $2DKP$ is solved using CPLEX version 12.6.1. The computation of this upper bound set is restricted to the considered triangle, following the method explained in (Delort and Spanjaard, 2010), i.e. a direction is investigated if and only if it is possible for the image of the optimal solution to be located in the explored triangle. Moreover, as it is done in (Delort and Spanjaard, 2010), the computation of this upper bound set at a given node is initialized with the supported efficient solutions of the relaxation of the parent node, which are feasible for the considered child node.

The second upper bound set is the Optimal Convex Surrogate Upper Bound set (OCSUB) defined in Chapter 4, (using the initialization by the dichotomic method). To build this upper bound set in a given triangle, the *0M-M1 interval* algorithm might use solutions whose image is located outside of the triangle. Thus the computation of the OCSUB can hardly be reduced to the investigated triangle. Two versions are considered in this chapter. The first one consists in computing the whole OCSUB at each node of the branch-and-bound algorithm, which might be expensive. The second version restrict each CSUB to the considered triangle, as done for the convex relaxation. The obtained upper bound set will not be the OCSUB restricted to the triangle since there is no guarantee to find all solutions associated to CSUBs. However, the computational time of this approximated version of the OCSUB should be lower than for the exact version. In the following, we call *ApproxOCSUB* the approximation version of the OCSUB.

For both versions, the set of critical multipliers at a node is initialized with the critical multipliers found during the computation of the upper bound set for the parent node, for which at least one associated solution is feasible for this child node.

The upper bound set based on the LP relaxation is restricted to the explored triangle, two constraints are added: the first ensures that the first objective value of a solution is greater than the one of the nadir point of this triangle; the second is similar for the second objective function. In order to avoid numerical instabilities during the evaluation of the different strategies, we have chosen to solve the LP relaxation using a dichotomic method (on the weighted sum of the objectives), instead of a parametric simplex. For each direction, the continuous single-objective bi-dimensional problem is solved using the simplex implementation of CPLEX.

Experimental results

In this section, we compare the upper bound sets described previously, i.e. the convex relaxation, the two versions of the surrogate relaxation (*OCSUB* and *ApproxOCSUB*) and the LP relaxation. The applied branching strategy fixes the variable in the same order as in the instance file. Since the instances are generated randomly, this strategy can be considered as a deterministic random strategy (*random*). No initialization of the lower bound set is done.

Table 5.1 presents the obtained results. Two indicators are used: the size of the search-tree (*nb nodes*), which indicates the quality of the upper bound set; and the computational time in seconds (*time*).

Table 5.1 is vertically separated in three parts corresponding to the size of the instances (from top to bottom: 28, 100 or 105, 150 variables). The variant giving the smallest search-tree for each instance is indicated in bold and blue and the one giving the smallest computational time is indicated in bold and green. The execution of the solution method is limited to one hour, if this limit is exceeded "-" appears instead of the size of the search-tree and the computational time. The columns are named according to the used relaxation.

instances	Convex relaxation		OCSUB		ApproxOCSUB		LP	
	nb nodes	time	nb nodes	time	nb nodes	time	nb nodes	time
A1	**780**	40.102	1172	389.191	1172	219.705	35670	**5.483**
A2	**1195**	16.006	2499	86.295	2647	43.357	12617	**2.472**
A3	**903**	36.765	1683	249.168	1693	132.758	42271	**7.596**
A4	**1669**	8.056	2455	49.966	2481	48.940	7115	**1.777**
D1	**700**	19.848	702	9.418	706	9.348	7922	**1.474**
D2	1538	9.087	**1528**	7.274	1538	13.341	7296	**1.604**
D3	921	11.536	**893**	10.849	913	6.204	19671	**3.187**
D4	**879**	8.433	**879**	4.477	**879**	7.310	11033	**1.846**
kp28	**2123**	8.332	2705	54.174	2737	33.646	7553	**1.604**
kp28-2	**202**	4.340	250	15.545	252	10.958	1734	**0.391**
kp28W-Perm	**587**	6.512	819	89.550	819	33.523	2803	**0.632**
kp28W-ZTL	**7**	0.492	39	1.922	39	2.051	175	**0.064**
kp28W	**708**	4.947	962	36.402	962	29.467	2698	**0.585**
kp28c1W-c2ZTL	**291**	2.796	705	9.636	747	11.794	2567	**0.532**
kp28cW-WZTL	**4102**	60.932	7674	91.804	7878	99.047	34686	**6.344**
ZTL28	**249**	4.219	373	14.225	395	11.290	3125	**0.695**
W7BI-rnd1-1800	**9376**	584.149	-	-	46288	2103.550	314304	**87.362**
W7BI-rnd1-3000	**2057**	41.637	10773	1041.190	11421	670.488	57005	**15.474**
W7BI-tube1-1800	**16524**	776.756	-	-	52554	3546.080	303730	**84.183**
W7BI-tube1-3000	**2968**	126.216	6250	251.006	6484	227.189	44562	**10.466**
W7BI-tube1-asyn	13093	250.563	**12975**	75.954	14493	125.583	115365	**36.440**
W7BI-tube2-1800	**17179**	742.130	-	-	-	-	332003	**87.574**
Wcollage-tube	**20065**	1007.720	-	-	-	-	795817	**217.923**
ZTL100	**3654**	539.400	17126	1592.790	20878	1345.840	441828	**118.809**
ZTL105	**5042**	973.930	-	-	45402	2492.440	696664	**228.764**
ZTL150-f250	-	-	-	-	-	-	**2986617**	**1003.960**
ZTL150-f500	-	-	-	-	-	-	**4841105**	**1745.780**

Table 5.1: Impact of the upper bound set on the performances of the branch-and-bound embedded in a two phase method to solve $2O2DKP$.

Regarding the size of the search-tree indicator, the results are as expected. Since the convex relaxation gives the tightest possible convex upper bound set, the variant using this relaxation generally leads to the smallest search-tree (the branches are cut earlier in the execution). However, for three instances the variant using the surrogate relaxation (OCUSB) leads to smaller search-tree than the one obtained by the variant using the convex relaxation. This can be explained by the fact that the upper bound set based on the surrogate relaxation is not restricted to the explored triangle, so it is possible to find during the exploration of the triangle a new feasible solution located in an other triangle. This improves the lower bound set of the latter and may allow to cut earlier in the search-tree during its exploration.

The use of the LP relaxation leads to largely bigger search-trees. However, the computation of this upper bound set is largely faster than the ones based on the convex or surrogate relaxation. This is why the LP relaxation leads to the smallest computational time for all instances.

The computational time obtained when using *ApproxOCSUB* as upper bound set is significantly lower than the one obtained for *OCSUB*, while only a slight increase of the size of the search-tree can be observed. *ApproxOCSUB* allows to solve two more instances than *OCSUB*, without exceeding one hour of execution. However, the computational times obtained for variants using the two upper bound sets based on the surrogate relaxation (*OCSUB* and *ApproxOCSUB*) are considerably larger than for the convex and the LP relaxation.

Consequently to the results observed in Table 5.1, in the remaining of this chapter, we use the upper bound set based on the LP relaxation.

5.2.3 Comparison of the branching strategies

As observed in Chapter 3, the branching strategies also play an important role in the practical efficiency of the method. In this section, we compare three branching strategies for the version of the method using the LP relaxation (since it is the one leading to the best computational time).

The first branching strategy is the one applied in the previous section, for all four versions of the algorithm. It is called *random*.

The two next strategies are based on the extreme efficient solutions of the LP relaxation. Since the efficient solutions of the LP relaxation are generally fractional solutions, branching on a variable which is "very" fractional in the upper bound set is interesting. Indeed the upper bound sets of the child nodes will be very different from the one of the parent node. The difficulty is to determine which variable is the "most" fractional in the LP relaxation. We have tested two slightly different measures, for a variable j: the number of extreme efficient solutions of the LP relaxation for which the variable j is fractional (*nbFrac*) and the distance of the value of the variable j to a integer value, summed up over all the extreme efficient solutions of the LP relaxation (*sumFrac*). The two branching strategies based on those measures fix the variables in decreasing order of the considered measure. An example of these two branching strategies is presented in Example 9.

Example 9. *Let us consider the $2O2DKP$ instance of Example 5 (Page 96):*

$$
\begin{array}{lrl}
\max & 10\,x_1 + 7\,x_2 + 20\,x_3 + 7\,x_4 + 8\,x_5 & \\
\max & 15\,x_1 + 17\,x_2 + 7\,x_3 + 4\,x_4 + 10\,x_5 & \\
\textit{s.t.} & 3\,x_1 + 1\,x_2 + 9\,x_3 + 4\,x_4 + 9\,x_5 \leq 13 & \qquad (2O2DKP\text{-}1)\\
& 13\,x_1 + 11\,x_2 + 2\,x_3 + 1\,x_4 + 7\,x_5 \leq 17 & \\
& x_j \in \{0,1\},\ j = 1\ldots 5 &
\end{array}
$$

The extreme efficient solutions of the LP relaxation are $(0.265, 0, 0, 0.449, 0)$ *and* $(0.261, 0, 0.198, 0, 0)$ *(rounded to* 10^{-3}*). Table 5.2 presents the value obtained for each variables for the two branching strategies* nbFrac *and* sumFrac.

Strategy	Measure value				
	x_1	x_2	x_3	x_4	x_5
nbFrac	2	0	1	1	0
sumFrac	0.526	0	0.198	0.449	0

Table 5.2: Measure value obtained for the two branching strategies *nbFrac* and *sumFrac*

The variable x_1 is thus chosen by the two strategies for the separation.

Experimental results

Table 5.3 analyzes the impact of the branching strategies when using the LP relaxation. The three presented branching strategies are compared over the two indicators: the size of the obtained search-tree (*nb nodes*) and the computational time (*time*). The considered instances are the same as in Section 5.2.2 and the table is split in three parts according to the size of the instances.

Table 5.3 shows that the branching strategy has an important influence on both the size of the search-time and the computational time. Indeed when the branching strategies are based on an analysis of the extreme efficient solutions of the LP relaxation, then the two indicators are reduced by around a factor 2 in average. Those two separation strategies, *nbFrac* and *sumFrac*, gives similar results. However *nbFrac* is slightly more efficient in average than *sumFrac*. Thus from now on we will only consider the separation strategy *nbFrac* when using the LP relaxation.

instances	random		nbFrac		sumFrac	
	nb nodes	time	nb nodes	time	nb nodes	time
A1	35670	5.483	**16936**	**3.092**	17158	3.102
A2	12617	2.472	**9537**	**2.140**	10187	2.292
A3	42271	7.596	**19177**	**3.791**	20165	3.960
A4	7115	1.777	4981	1.182	**4725**	**1.130**
D1	7922	1.474	3740	0.907	**3710**	**0.882**
D2	7296	1.604	**3754**	**0.917**	4058	0.978
D3	19671	3.187	**3899**	**0.911**	4645	1.015
D4	11033	1.846	**4517**	**0.944**	5341	1.071
kp28	7553	1.604	**3201**	**0.748**	4001	0.907
kp28-2	1734	0.391	**722**	**0.258**	882	0.286
kp28W-Perm	2803	0.632	**1291**	**0.444**	1539	0.513
kp28W-ZTL	175	0.064	**123**	**0.058**	147	0.062
kp28W	2698	0.585	**1112**	**0.360**	1278	0.387
kp28c1W-c2ZTL	2567	0.532	**1267**	**0.315**	1283	0.323
kp28cW-WZTL	34686	6.344	**19432**	**4.342**	20948	4.584
ZTL28	3125	0.695	**1635**	**0.452**	1715	0.460
W7BI-rnd1-1800	314304	87.362	**66434**	**24.299**	106146	37.539
W7BI-rnd1-3000	57005	15.474	**7167**	**2.621**	8805	3.026
W7BI-tube1-1800	303730	**84.183**	**247820**	91.904	430842	147.777
W7BI-tube1-3000	44562	10.466	**14060**	**4.927**	17878	5.856
W7BI-tube1-asyn	115365	36.440	**91021**	**29.528**	169497	58.230
W7BI-tube2-1800	332003	87.574	**244397**	**83.802**	510871	167.853
Wcollage-tube	795817	217.923	**64577**	**26.904**	82415	32.408
ZTL100	441828	118.809	186258	69.166	**179970**	**66.938**
ZTL105	696664	228.764	**289664**	**122.272**	310520	129.256
ZTL150-f250	2986617	1003.960	**1011437**	**433.879**	1060527	458.828
ZTL150-f500	4841105	1745.780	**1434267**	**676.123**	1440927	676.563

Table 5.3: Impact of the branching strategy on the performances of the branch-and-bound embedded in a two phase method to solve $2O2DKP$, when the upper bound set is based on the LP relaxation.

5.2.4 Initialization of the algorithm using a path relinking

In this section we want to evaluate the benefit of using a procedure initializing the branch-and-bound method with feasible solutions. The aim of the initialization is to find "high" quality solutions, which will allow us to fathom nodes from the early iterations of the algorithm. Since the solving is done by a two phase method, supported efficient solutions are found before we launch a branch-and-bound in each triangle (before the second phase). These solutions can be used to generate other ones of good quality. It has been done in (Gandibleux et al., 2001), for example.

Our initialization method consists in applying several times a path relinking operator between pairs of supported efficient solutions. The path relinking operator ((Glover et al., 2000), (Glover and Kochenberger, 2003) for further details on this method) builds a path between two solutions, called the initiating and the guiding solutions. It is initialized with a initiating solution and seeks solutions to reduce the distance to a guiding solution, according to a distance measure.

The pseudo-code of our initialization method is given in Algorithm 6.

The initialization method executes the same number of calls to the path relinking operator as the number of triangles defined by the supported efficient solutions. The initiating and guiding solutions of an execution of the path relinking operator are chosen at Line 4. A path relinking is executed at each iteration of Lines 5-13. Each step of the path relinking operator (each iteration of the loop at Lines 6-10) generates a feasible solution x such that the Hamming distance between the guiding solution and x (i.e. the number of bits differing between to binary solutions) is less than

Algorithm 6: InitializationByPathRelinking

input : A $2O2DKP$ instance
SE: the set of supported efficient solutions
output: SF: the set of feasible solutions found

1 **begin**
/* ChooseInitGuid (nbIter↓, SE↓, initSol↑, guidingSol↑) chooses the initiating and guiding solutions (initSol and guidingSol) among the supported efficient solutions in SE at the iteration nbIter */
/* HD(x^1↓, x^2↓) gives the Hamming distance between x^1 and x^2 */
/* Neighborhood(x↓) returns the neighborhood of x */
/* ChooseNeighbor(feasibleNeighbors↓) chooses the solution to continue the path, among the neighborhood feasibleNeighbors */
/* Completion(x↓) is the completion mechanism applied on x */
2 nbIter ← 0
3 **while** nbIter < |SE| − 1 **do**
4 ChooseInitGuid (nbIter↓, SE↓, initSol↑, guidingSol↑)
5 x ← initSol
6 **while** $x \neq$ guidingSol **do**
7 feasibleNeighbors ← $\{x' \in$ Neighborhood(x↓) : x' is feasible and HD(x'↓, guidingSol↓) < HD(x↓, guidingSol↓)$\}$
8 x ← ChooseNeighbor(feasibleNeighbors↓)
9 SF ← SF ∪ $\{x\}$
10 **if** x *is not complete* **then**
11 sols ← Completion(n)
12 SF ← SF ∪ sols
13 nbIter ← nbIter + 1
14 **return** SF

the Hamming distance between the guiding solution and x' the solution generated at the previous iteration. The solution x is chosen in the neighborhood of x', i.e. the set of solutions differing from x' by only one bit (Line 7 and 8). Since we aim to generate feasible solution of good quality, only the feasible solution in the neighborhood of x' are considered. The path relinking stops when the solution x added to the path is the guiding solution.

Among the solutions along the path, we can distinguish two categories, based on the saturation of the capacities: the complete and the non-complete solutions (Definition 26).

Definition 26. *A solution is called* complete *if it is not possible to add an item without exceeding at least one capacity constraint.*

Obviously, a non-complete solution cannot be efficient. Indeed, the value of the objective function can be improved by adding another item. The mechanism at Lines 10 to 12, called the completion, aims to generate better solutions than the non-complete solutions of the path, by selecting additional items. The solutions generated by this mechanism are not part of the path. To increase the chance to find good solutions, we complete a non-complete solution in any possible ways (adding every valid combination of objects).

The solutions from the path and from the completion compose the set of feasible solutions returned by the initialization, filtered by dominance.

Example 10 gives a numerical example of the path relinking initialization.

Example 10. *Let us consider the following* $2O2DKP$ *instance:*

$$
\begin{array}{lrcl}
\max & 180\,x_1 + 80\,x_2 + 90\,x_3 + 200\,x_4 + 110\,x_5 & & \\
\max & 29\,x_1 + 128\,x_2 + 127\,x_3 + 16\,x_4 + 112\,x_5 & & \\
\textit{s.t.} & 180\,x_1 + 76\,x_2 + 90\,x_3 + 198\,x_4 + 98\,x_5 & \leq & 423 \\
& 13\,x_1 + 99\,x_2 + 112\,x_3 + 55\,x_4 + 60\,x_5 & \leq & 167 \\
& x_j \in \{0,1\},\ j = 1 \ldots 5 & &
\end{array}
\qquad (2O2DKP\text{-}2)
$$

The supported efficient solutions are:
– $(1,0,0,1,0)$ *of value* $(380, 45)$
– $(0,0,0,1,1)$ *of value* $(310, 128)$
– *and* $(0,1,0,0,1)$ *of value* $(190, 240)$.

Thus two path relinking executions will be done during the initialization. Any pair of supported efficient solutions can be chosen to constitute the initiating and guiding solutions. Let us consider $(1,0,0,1,0)$ *as initiating solution and* $(0,1,0,0,1)$ *as guiding solution.*

Table 5.4 presents the execution of the initialization.

(1) It	(2) x	(3) *feasibleNeighbor*	(4) chosen	(5) complete?	(6) completions
1	$(1,0,0,1,0)$	– $(0,0,0,1,0)$ – $(1,0,0,0,0)$	$(0,0,0,1,0)$	no	– $(1,0,0,1,0)$ – $(0,1,0,1,0)$ – $(0,0,1,1,0)$ – $(0,0,0,1,1)$
2	$(0,0,0,1,0)$	– $(0,1,0,1,0)$ – $(0,0,0,0,0)$ – $(0,0,0,1,1)$	$(0,1,0,1,0)$	yes	
3	$(0,1,0,1,0)$	$(0,1,0,0,0)$	$(0,1,0,0,0)$	no	– $(1,1,0,0,0)$ – $(0,1,0,1,0)$ – $(0,1,0,0,1)$
4	$(0,1,0,0,0)$	$(0,1,0,0,1)$			

Table 5.4: Execution of the initialization with the initiation solution $(1,0,0,1,0)$ and the guiding solution $(0,1,0,0,1)$. Column (1) numbers the executions of the Lines 6 to 13 of Algorithm 6, (2) presents the solution x considered at these lines, (3) the neighborhood obtained at Line 7, (4) indicates the solution we suppose to be chosen at Line 8, (5) indicates if the solution chosen if complete and (6) gives the solutions obtained from the completion mechanism Lines 11 and 12 if it is applied.

We can remark that the third solution from the completion in the first iteration selects an item (x_3) *which is neither in the guiding or the initiating solution.*

At the iteration 4, feasibleNeighbors *is composed only of the guiding solution. Thus the initialization stops.*

The set of feasible solutions, filtered by dominance, returned by the initialization method is given in Table 5.5.

Solution	Value
$(1,0,0,1,0)$	$(380,45)$
$(0,0,0,1,1)$	$(310,128)$
$(0,0,1,1,0)$	$(290,143)$
$(0,1,0,1,0)$	$(280,144)$
$(1,1,0,0,0)$	$(260,157)$
$(0,1,0,0,1)$	$(190,240)$

Table 5.5: Feasible solutions returned by the initialization method

All the solutions in Table 5.5 are efficient solution and the only missing efficient solution is $(1,0,1,0,0)$ *of value* $(270,156)$.

We can see in Example 10 that there are two main points of choice in the initialization method: the choice of the initiating and guiding solutions (Line 4 of Algorithm 6) and the choice of the next neighbor in the path (Line 8 of Algorithm 6). In the next paragraph, we propose two implementations for the choice of the initiating and guiding solutions and three implementations for the choice of the neighbor.

Choice of the initiating and guiding solutions and choice of the neighbor

Choosing different pairs of initiating and guiding solutions lead to different paths and thus different feasible solutions. The two most intuitive choice strategies are: choosing randomly a pair of supported solutions and choosing adjacent supported solutions. The first strategy, called *randSol*, favors the diversity of the solution. The second one, named *adjSol*, aims to find points in the triangle defined by the two considered supported efficient solution.

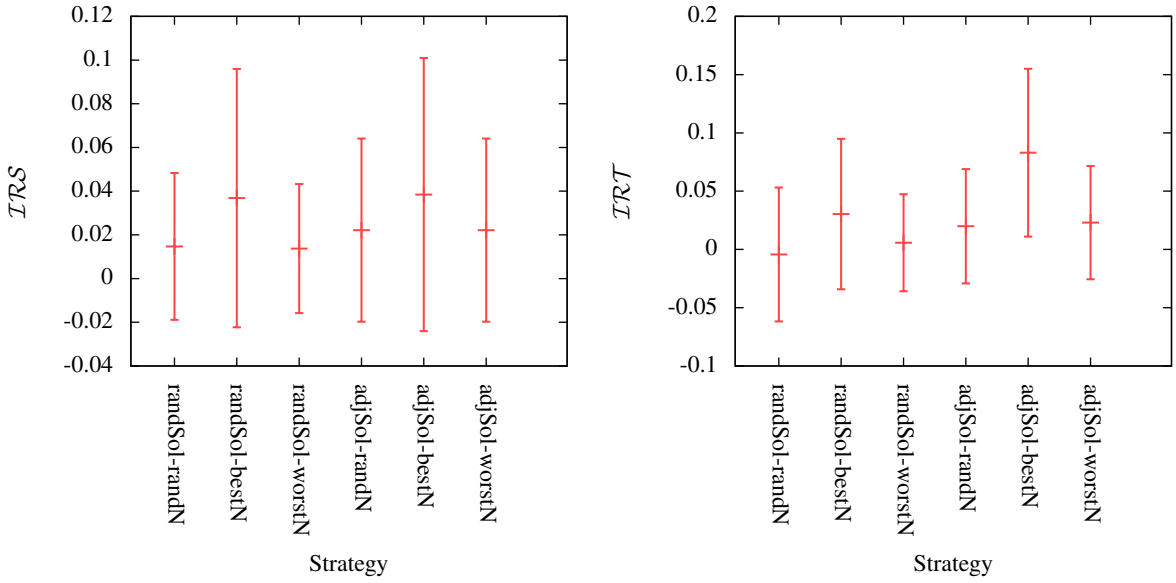
Selecting the solution to add in the path, at a given iteration, can be done according to different strategies. In particular, the selected solution can be the one maximizing the weighted sum defined by the initiating and guiding solutions (named *bestN*), or minimizing this measure (named *worstN*); or it can be the result of a random choice (named *randN*).

Those variants on the choice of the initiating and guiding solution and on the choice of the neighbor define six different implementations for the path relinking initialization. The six initialization versions are evaluated by launching the two phase method whose second phase is a branch-and-bound method, using the tested initialization, the LP relaxation to compute the upper bound sets and the *nbFrac* branching strategy. The name of an initialization version is the juxtaposition of its components. The instances are presented in Section 5.1 with 28, 100 or 105 variables. Larger instances are not computed to avoid large computational times. The obtained performances are compared to the non-initialized method, on two indicators: the size of the search-tree and the computational time.

The first indicator, on the size of the search-tree, aims to evaluate the quality of the solutions found. Indeed a better quality of the solutions implies a smaller search-tree. The measure used for this indicator is the improvement ratio $\mathcal{IRS}$, defined for a given instance as: $\mathcal{IRS} = \frac{s^r - s^t}{s^r}$ where s^r is the size of the search-tree obtained by the reference method (non-initialized) and s^t is the size of the search-tree obtained by the method using the tested initialization. Figure 5.1a shows the average and standard deviation of this measure over all instances.

Finding good solutions has a cost and even if the search-tree is largely decreased, the computational time can be deteriorated. The second indicator, comparing the computational times, uses

the measure $\mathcal{IRT}$, defined for a given instance as: $\mathcal{IRT} = \frac{t^r - t^t}{t^r}$ where t^r is the computational time spent by the reference method (non-initialized) and t^t is the computational time spent by the method using the tested initialization. Figure 5.1b shows the average and standard deviation of $\mathcal{IRT}$ for the considered instances.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 5.1: Comparison of the six variants of the path relinking initialization for two phase method whose second phase is a branch-and-bound method. The non-initialized version of the method is the reference.

On Figure 5.1, we can observe that building the path using the strategy *BestN* leads in average to better performances than the two others strategies, independently of the choice of initiating and guiding solutions. This strategy is the best regarding the computational time and the size of the search time. The two other strategies (*randN* and *WorstN*) seem to have equivalent performances.

We can also remark that choosing adjacent solutions as initiating and guiding solutions leads to smaller search-trees and smaller computational times in average than choosing the initiating and guiding solutions randomly.

Thus the strategy giving the smallest search-tree and the smallest computational time is the one executing a path relinking between two adjacent supported solutions (*adjSol*) and choosing the best neighbor to built the path among feasible neighbors (*bestN*). In the following, we use this variant.

Oscillation

As already mentioned, the solutions along the path can be divided in two categories: the complete and non-complete solutions. The non-complete solutions cannot be efficient. However, an improvement mechanism is applied on them (the completion Line 10 to 12 of Algorithm 6). Thus the non-complete solutions of the path allow to generate several solutions.

On the contrary, the complete solutions do not generate additional solutions. Indeed the completion mechanism cannot be applied on a complete solution.

On another hand, the solutions found during the path relinking executions, are quite similar and it may be in particular the case for the solutions generated by the completion procedure. However the efficient solutions can be very diversified. Therefore, it might be interesting to exploit the complete solutions to generate several diversified feasible solutions.

To diversify the feasible solutions, we introduce a last mechanism called *oscillation* (Hanafi and Fréville, 1998), that we add between Lines 9 and 10 of Algorithm 6, if the solution is complete. Its principle is to degrade a complete solution $\bar{x}$ by adding items (making it infeasible), then the solution is restored generating several feasible solutions.

The first phase, consisting in adding items making the solution infeasible is called the *degradation*. During this phase, a given number of items, with the value 0 in $\bar{x}$, are set to 1. We call the *depth* of the degradation the number of items added during the degradation phase, it is a parameter of the procedure. In order to obtain good feasible solutions from the oscillation mechanism, the items are added in decreasing order of the utility $u_j^\lambda = \dfrac{\lambda_1 z_j^1 + \lambda_2 z_j^2}{w_{1j} + w_{2j}}$, where λ is defined as the normal of the hypothenuse of the triangle formed by the initiating and guiding solutions (since the strategy *adjSol* is employed). The solution obtained at the end of the degradation phase is denoted by x^D.

The second phase, called the restoration phase, consists in removing items from x^D, which is infeasible, in order to obtain feasible solutions. There exist several ways to restore the solution x^D, depending on the set of items removed from the solution. We can apply a similar principle than the completion, by generating every possible restorations. However, we are interested only in good quality feasible solutions, therefore we generate only complete solutions (otherwise there are proven to be dominated). The restoration procedure can then be represented by a decision-tree in which a path corresponds to a set of deleted items and a node is pruned whenever the corresponding solution is feasible. A node can also be pruned when the corresponding solution is dominated by a supported solution, indeed in this case the solution cannot improve the lower bound set.

Example 11 presents an example of the oscillation procedure.

Example 11. *Let us consider the problem $2O2DKP$-2, presented in Example 10, and the solution $\bar{x} = (0,1,0,1,0)$ obtained at the iteration 2 of the initialization procedure in Example 10.*

We apply an oscillation with a depth of degradation of 1, in the path relinking presented in Example 10. Then $\lambda = (240-45, 380-190) = (195, 190)$. The item, with a value of 0 in $\bar{x}$, with the highest utility u^λ is x_5. Thus the solution x^D is $(0,1,0,1,1)$. The restoration phase builds the decision-tree presented in Figure 5.2. The solutions generated by the restoration phase are $(0,1,0,0,1)$, $(0,0,0,1,1)$ and $(0,1,0,1,0)$, their respective value are $(190,240)$, $(310,128)$ and $(280,144)$. The other possible restorations of x^D would give non-complete solutions.

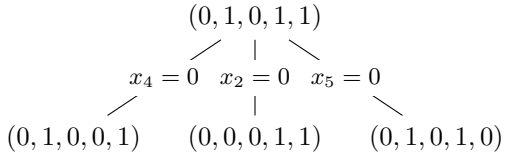


Figure 5.2: Decision-tree for the restoration on the solution $x^D = (0,1,0,1,1)$

If a depth of degradation is 2, then the solution x^D would be $(1,1,0,1,1)$, the decision-tree built is presented in Figure 5.3. The solutions generated by the restoration phase would

be $(0,1,0,0,1)$,$(1,0,0,0,1)$, $(1,1,0,0,0)$, $(0,0,0,1,1)$, $(0,1,0,1,0)$ *and* $(1,0,0,1,0)$*, their respective value are* $(190,240)$, $(290,141)$, $(260,157)$, $(310,128)$, $(280,144)$ *and* $(380,45)$.

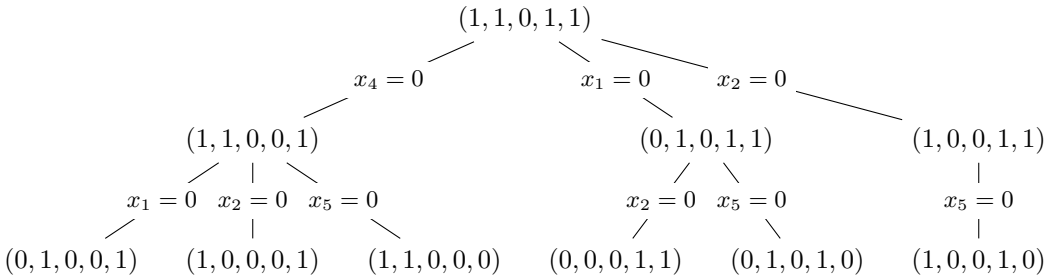


Figure 5.3: Decision-tree for the restoration on the solution $x^D = (1,1,0,1,1)$

As we can see in Example 11, the quality of the solutions obtained by the oscillation mechanism depends on the depth of the degradation. Indeed when the depth of the degradation increases the quality and the number of obtained solutions increase too, but the computational time also increases. In order to limit the computational time spent, while keeping a satisfying quality for the generated solutions, we introduce a maximal number of solutions generated by the restoration. Then the procedure favors the deletion of items j having the lowest u_j, in the aim of maintaining the quality of the feasible solutions. This parameter is illustrated in Example 12.

Example 12. *In Example 11, when the depth is 2, if the maximum number of solutions from the restoration phase is 3, then the first item to be deleted is* x_4 *since it has the lower* u^λ *with* $\lambda = (195,190)$*. Then only the solutions* $(0,1,0,0,1)$, $(1,0,0,0,1)$ *and* $(1,1,0,0,0)$ *would be generated during the restoration phase.*

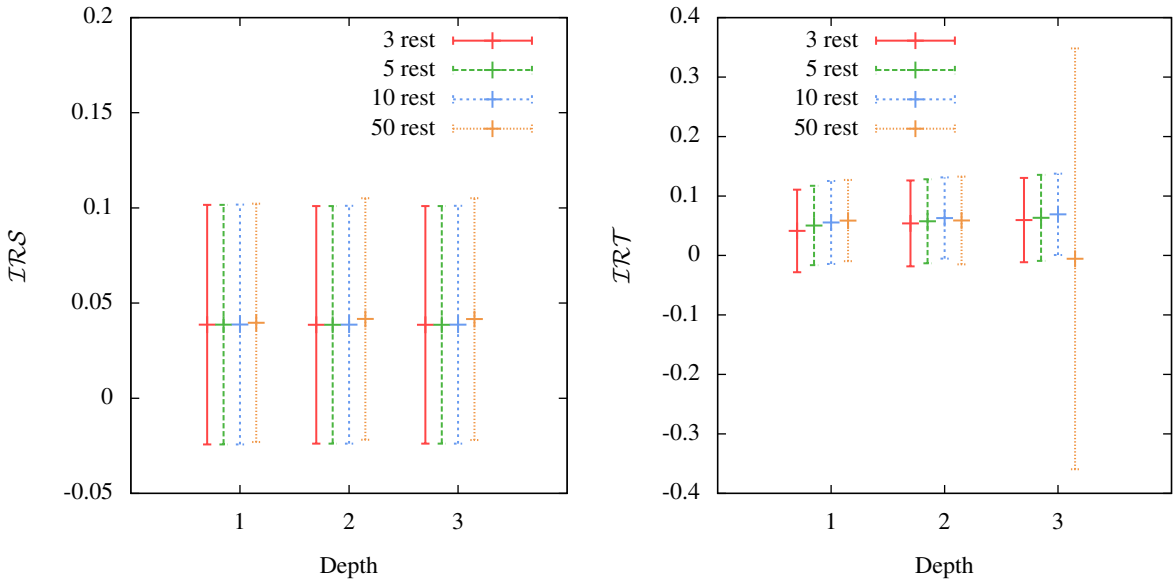
The impact of the depth of the degradation, as well as the impact of the maximum number of restored solutions generated are tested on instances of 28, 100 and 105 variables. The method executed is the branch-and-bound method embedded in a two phase method, using the LP relaxation and the *nbFrac* branching strategy. The initialization uses the strategies *adjSol* and *bestN*. Figure 5.4 shows the obtained results, with respect to different depths of the degradation; "n rest" specifies the maximum number of restored solutions generated from a complete solution.

Figure 5.4a confirms that when the depth and the number maximum of solutions from the restoration increase, the quality of the initial solutions increases and thus the size of the search-tree decreases. However Figure 5.4b shows that the computational time increases in this case. The depth of the degradation does not impact significantly the size of the search-tree for a fixed maximum number of restorations. To get a good tradeoff between the quality of obtained solutions and the computational time during the oscillation procedure, it is important to reduce the maximum number of solutions generated during the restoration phase. The optimum maximum number of generated solutions during this phase seems to be 5.

Considering the results in Figure 5.4, we will apply the initialization procedure with a degradation of depth 2 and a maximum number of solutions from the restoration of 5.

Restriction of the number of completions

The principle to generate only a restricted number of solutions during the restoration phase (presented in the previous paragraph) can be adapted to the completion procedure (Line 10 to 12 of Algorithm 6). In order to evaluate the impact of the maximal number of solutions generated by



(a) According to the size of the search-tree

(b) According to the computational time

Figure 5.4: Impact of the depth of the degradation and of the maximal number of solutions generated during the restoration phase

the completion procedure on the initialization method, we run the method initialized, with different values for this parameter. The initialization uses the strategies *adjSol*, *bestN*, degradations of depth 2 and restorations with a maximum number of solutions of 5. The upper bound set of the branch-and-bound method is based on the LP relaxation and the branching strategy is *nbFrac*.

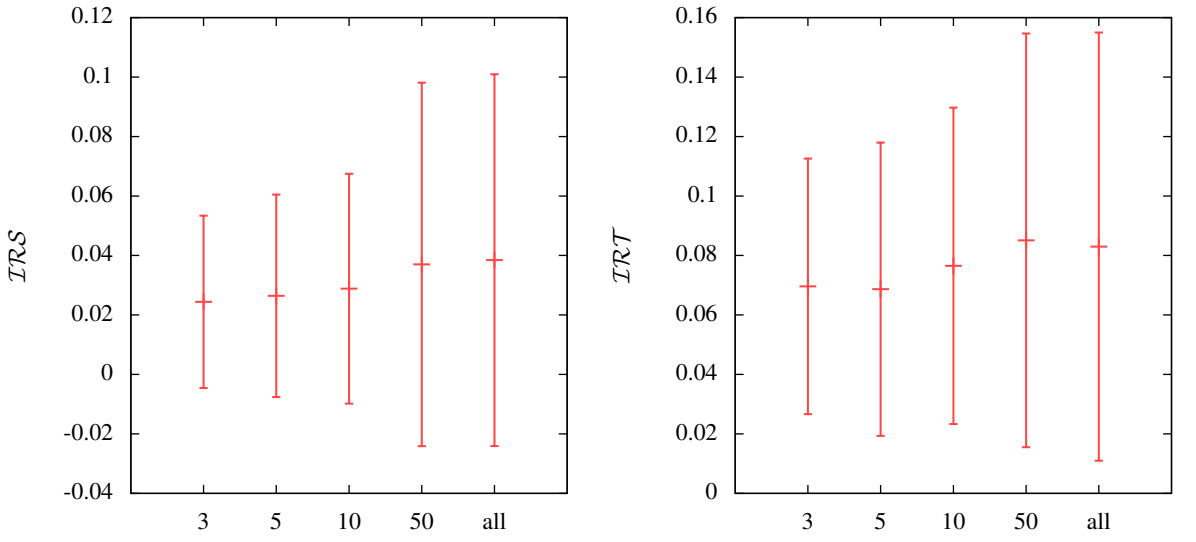
Figure 5.5 shows the impact of this maximum number of solutions generated during the completion procedure, regarding the size of the search-tree and the computational time, on instance of 28, 100 or 105 variables.

Figure 5.5a shows that, as expected, when the maximum number of solutions generated by the completion procedure increases, the size of the search-tree decreases ($\mathcal{IRS}$ increases). However, when the maximum number of solutions generated by the completion procedure increases, $\mathcal{IRT}$ increases too, except when more than 50 solutions are allowed. However the difference in terms of $\mathcal{IRT}$ is rather small when comparing the different tuning. In the following a maximum number of 50 solutions is generated by the completion procedure.

5.2.5 Initialization by the nondominated set

Even if the path relinking initialization allows to reduce the size of the search-trees and the computational time in average, this reduction is modest. Indeed the size of the search-tree is reduced, on average, by approximately 4% and the computational time by approximately 8%. In order to evaluate the maximum reduction we can reach with an initialization method, we initialize the feasible solutions with the set of efficient solutions. Obviously this initialization is an ideal initialization, which could not be achieved in practice. However since this is the best possible initialization, it gives an upper bound on the reduction of the size of the search-tree and the one of the execution time.

Figure 5.6 compares the performance obtained for the branch-and-bound based method with

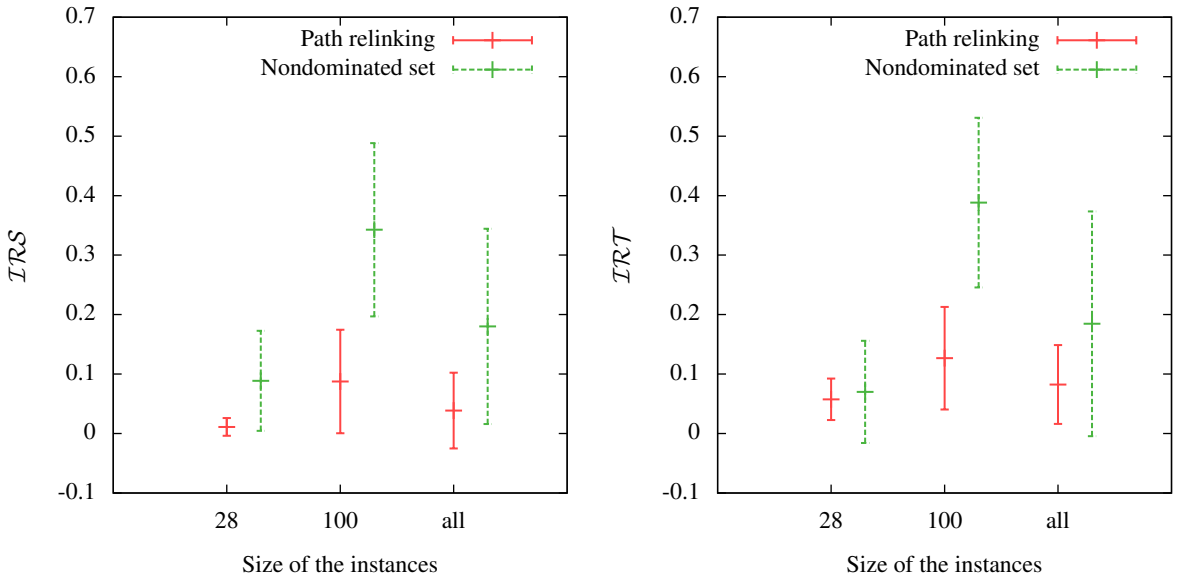


(a) According to the size of the search-tree

(b) According to the computational time

Figure 5.5: Impact of the maximal number of completion for each non-complete solution

the initialization method by the path relinking to the one using the initialization by the nondominated set. The path relinking initialization uses the strategies *adjSol* and *bestN*, with a degradation of depth 2, 5 solutions generated during the restoration and 50 during the completion.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 5.6: Comparison of the initialization using the path relinking and the initialization by the nondominated set.

Figure 5.6a shows that initializing the method with the nondominated set allows to reduce the

search-tree by 16% in average. The difference of $\mathcal{IRS}$ seems to grow when the size of the instance increases.

Regarding the computational time (Figure 5.6b), the initialization with the nondominated set allows a reduction of 18%. The path relinking initialization allows almost half the possible reduction.

The average reduction of the computational time is highly correlated with the average reduction of the size of the search-tree, which indicates that the cost of the path relinking initialization is reasonable. There still exists a large margin of improvement for the feasible solutions obtained by the initialization method. However finding good solutions can be expensive. The path relinking initialization method seems to provide an interesting tradeoff between the quality of the obtained feasible solutions and the computational time spent on the initialization.

From the results presented in Figure 5.6, we can also observe that the reduction of the size of the search-tree is modest when the method is initialized with the nondominated set. Thus the upper bound set might not be tight enough and does not allow to prune branches early in the execution. In Section 5.4, we introduce valid inequalities all along the solving process, in order to tighten the upper bound set and thus prune more branches. The valid inequalities introduced are cover inequalities, which are presented, in the single-objective context, in the next section.

5.3 Cover inequalities for the multi-dimensional knapsack problem

As observed in Table 5.1, the LP relaxation leads to a low computational time, however the size of the search-trees obtained for this relaxation is largely higher than the one obtained for the other relaxations. The idea behind the branch-and-cut method embedded in a two phase method is to tighten the upper bound set obtained using the LP relaxation by adding valid inequalities. In this section, we present the cover inequalities, which are valid inequalities introduced for the $mDKP$, and we present two problems leading to cover inequalities based on the optimal solution of the LP relaxation.

The notion of *cover inequality* was introduced by Crowder et al. (1983). A set C of items j is a *cover* for the constraint i $\left(\sum_{j=1}^{n} w_{ij}\, x_j \leq \omega_i\right)$ if the sum of the weights of those items exceeds the capacity ω_i, $i \in \{1, \ldots, m\}$. The cover inequality corresponding to C is $\sum_{j \in C} x_j \leq |C| - 1$. A cover C is minimal if deleting any item of C implies that the resulting set is no longer a cover.

A cover inequality can be extended by adding all items which have a weight greater or equal to any item in the cover. We call $E(C)$ the extended set from cover C on the considered dimension. The resulting inequality is then $\sum_{j \in E(C)} x_j \leq |C| - 1$.

Since the definition of a cover inequality does not involve the objective function, the definition is the same for single-objective and multi-objective contexts.

Example 13. *We consider the* $2O2DKP$ *instance:*

$$
\begin{array}{lrcl}
\max & 10\,x_1 + 7\,x_2 + 20\,x_3 + 7\,x_4 + 8\,x_5 & & \\
\max & 15\,x_1 + 17\,x_2 + 7\,x_3 + 4\,x_4 + 10\,x_5 & & \\
\textit{s.t.} & 3\,x_1 + 1\,x_2 + 9\,x_3 + 4\,x_4 + 9\,x_5 & \leq 13 & \qquad (2O2DKP\text{-}1) \\
& 13\,x_1 + 11\,x_2 + 2\,x_3 + 1\,x_4 + 7\,x_5 & \leq 17 & \\
& x_j \in \{0,1\},\ j = 1, \ldots, 5 & &
\end{array}
$$

$x_2 + x_3 + x_4 \leq 2$ is a cover inequality (according to the first constraint). It can be extended in $x_2 + x_3 + x_4 + x_5 \leq 2$ since x_5 has a weight (9) greater or equal than x_2, x_3 and x_4 (respectively 1, 9 and 4).

In (Crowder et al., 1983), the authors remark that computing all cover inequalities would be very time-consuming and even impossible to implement. Instead they consider the optimal solution of the LP relaxation x^{LP} and aim to find a cover inequality violated by x^{LP}. To find such a cover, they solve the following binary linear program, defined on the constraint i. This problem is solved successively for every constraint.

$$\begin{aligned} \min & \sum_{j=1}^{n} (1 - x_j^{LP})\, z_j \\ s.t. & \sum_{j=1}^{n} w_{ij}\, z_j \geq \omega_i + 1 \\ & z_j \in \{0,1\} \qquad\qquad j = 1, \ldots, n \end{aligned} \tag{5.1}$$

We call z^* the optimal solution of Problem (5.1). The cover defined by z^* is the set of items j such that $z_j^* = 1$.

They highlight a condition (cf Proposition 13) under which the cover inequality found is violated by x^{LP}. In the following we will say that such a cover inequality is *cutting*.

Proposition 13. *(Crowder et al., 1983) The cover inequality defined by the optimal solution z^* of (5.1) is violated by x^{LP} if and only if the objective value of z^* is strictly lower than 1.*

They pointed out that Problem (5.1) finds the "most violated" cover inequality. Klabjan et al. (1998) prove that Problem (5.1) is a NP-hard problem.

Bektas and Oguz (2007) explain that Problem (5.1) can be reduced to the set of fractional variables for the LP relaxation, i.e. such that $0 < x_j^{LP} < 1$. To explain that, we split the set of variables according to their value in x^{LP}: $J_0 = \{j, x_j^{LP} = 0\}$, $J_1 = \{j, x_j^{LP} = 1\}$ and $J_F = \{j, 0 < x_j^{LP} < 1\}$.

According to Proposition 13, the cover inequality is cutting if and only if the objective value of z^* is strictly lower than 1. Thus if we have $z_j^* = 1$ for any $j \in J_0$, then the objective value of z^* would be greater or equal to one and the resulting cover inequality will not be violated by x^{LP}. Thus, we can systematically fix z_j^* to 0 for every $j \in J_0$. Moreover for any $j \in J_1$, the objective coefficient of z_j is 0. Thus there exists one optimal solution z^* with $z_j^* = 1$ for each $j \in J_1$. We can reduce the size of Problem (5.1) by fixing each variable of J_1 to 1. This preprocessing does not affect the value of the optimal solution.

Problem (5.1) can be reformulated as follow:

$$\begin{aligned} \min & \sum_{j \in J_F} (1 - x_j^*)\, z_j \\ s.t. & \sum_{j \in J_F} w_{ij}\, z_j \geq \omega_i - \sum_{j \in J_1} w_{ij} + 1 \\ & z_j \in \{0,1\} \qquad\qquad j \in J_F \end{aligned} \tag{5.2}$$

Since the original problem is a multi-dimensional knapsack, fixing z_j to 0 for each $j \in J_0$ can lead to an infeasible Problem (5.2) for one or several dimensions (but not all). In that case, we can deduce that no cover inequality based on this constraint allows to cut the solution x^{LP}. If

the solution x^{LP} is not integer, then the analysis of the other constraints will give a cutting cover inequality.

Since the objective coefficient of some variables are 0 in Problem (5.1), the cover obtained based on its optimal solution might not be minimal. However it is well known that minimal covers are tighter than non minimal ones.

To avoid that situation, Gabrel and Minoux (2002) propose to solve another binary problem with a non-linear objective function. In this chapter, the cover inequalities are minimalized by deleting the items with the smallest weight from the cover.

The use of Problems (5.1) and (5.2) is illustrated in Example 14.

Example 14. *We consider the weighed sum using multiplier (0,1) of the $2O2DKP$ instance presented in Example 13, i.e. the following $2DKP$ problem:*

$$\begin{array}{lrcl} \max & 15\,x_1 + 17\,x_2 + 7\,x_3 + 4\,x_4 + 10\,x_5 & & \\ \textit{s.t.} & 3\,x_1 + 1\,x_2 + 9\,x_3 + 4\,x_4 + 9\,x_5 & \leq & 13 \\ & 13\,x_1 + 11\,x_2 + 2\,x_3 + 1\,x_4 + 7\,x_5 & \leq & 17 \\ & x_j \in \{0,1\},\ j = 1 \ldots 5 & & \end{array} \qquad (2DKP\text{-}1)$$

The optimal solution of 2DKP-1 is $x^{LP} = (0.26, 1, 0.80, 1, 0)$, rounded to 10^{-2}. Problem (5.1) defined on x^{LP} based on the first constraint is:

$$\begin{array}{lrcl} \min & 0.74\,z_1 + 0\,z_2 + 0.2\,z_3 + 0\,z_4 + 1\,z_5 & & \\ \textit{s.t.} & 3\,z_1 + 1\,z_2 + 9\,z_3 + 4\,z_4 + 9\,z_5 & \geq & 14 \\ & z_j \in \{0,1\},\ j = 1, \ldots, 5 & & \end{array}$$

Fixing z_2 and z_4 to 1 and z_5 to 0, Problem (5.2) derived from Problem (5.1) is:

$$\begin{array}{lrcl} \min & 0.74\,z_1 + 0.2\,z_3 & & \\ \textit{s.t.} & 3\,z_1 + 9\,z_3 & \geq & 4 \\ z_j \in \{0,1\},\ j = 1,3 & & & \end{array}$$

The optimal solution of this reduced problem is $z_1 = 0, z_3 = 1$. Therefore, considering the first constraint leads to the cover inequality $x_2 + x_3 + x_4 \leq 2$. The same process considering the second constraint gives the cover inequality $x_1 + x_2 + x_4 \leq 2$.

We can remark that the cover inequality $x_1 + x_2 + x_4 \leq 2$ obtained using the second constraint is not minimal, since $\{x_1, x_2\}$ constitutes a cover for the second constraint.

5.4 Branch-and-cut for the bi-objective bi-dimensional knapsack problem

In this section, we aim to generate cover inequalities all along the solution method for $2O2DKP$. Since the relaxation used to compute the upper bound set is the LP relaxation, this section will only consider cover inequalities based on the LP relaxation. The main difference with the previous section is that the solved problem is a multi-objective problem, thus there does not exist one single optimal solution, but a set of efficient solutions for the LP relaxation. Problems (5.1) and (5.2), defined in the previous section, can be defined for any efficient solution of the LP relaxation.

In the first part of this section, we wonder if it is interesting to generate cover inequalities based on the surrogate constraint of $2O2DKP$. The second part presents the branch-and-cut method and different possible implementations for its components. The solution method implemented in this

method is based on the method elaborated in Section 5.2.1, i.e. the two phase method, whose second phase is a branch-and-bound. The branch-and-cut method uses all the components of this branch-and-bound method: the upper bound set is based on the LP relaxation; the branching strategy is *nbFrac* and the initialization method is the one defined in Section 5.2.4. Finally, the last part of this chapter presents experimental results obtained for the different implementations of the branch-and-cut method and determines the one leading to the best performances.

5.4.1 Cover inequalities and surrogate constraint

In this part of the chapter, we consider x^{LP} an efficient solution of the LP relaxation of $2O2DKP$.

We remark that Problems (5.1) and (5.2) are based on one constraint whereas $2O2DKP$ is composed of two constraints. By definition, Problems (5.1) and (5.2) can be defined for one or the other of the constraints, and by solving Problem (5.2) consecutively for both constraints, the obtained cover inequalities may be different (cf Example 14).

In this Section we wonder if analyzing an aggregation of the two constraints allows us to obtain tighter valid inequalities. The aggregation uses a multiplier $u \in [0,1]$, the obtained constraint, called the surrogate constraint of $2O2DKP$, is:

$$\sum_{j=1}^{n} u\, w_{1j}\, x_j + (1-u)\, w_{2j}\, x_j \leq u\, \omega_1 + (1-u)\, \omega_2$$

Any feasible solution of $2O2DKP$ also respects the surrogate constraint. Thus the surrogate constraint can be used to find cover inequalities without altering the feasible set. By defining Problem (5.1) with the surrogate constraint, we obtain:

$$\begin{aligned} &\min \sum_{j=1}^{n} (1 - x_j^{LP})\, z_j \\ &s.t. \sum_{j=1}^{n} u\, w_{1j}\, z_j + (1-u)\, w_{2j}\, z_j \geq u\, \omega_1 + (1-u)\, \omega_2 + 1 \\ &z_j \in \{0,1\} \qquad\qquad j = 1, \ldots, n \end{aligned} \tag{5.3}$$

We call z^0 and z^1 the optimal solutions of Problem (5.3) for respectively $u = 0$ and $u = 1$. Let us consider $\bar{z}$, the optimal solution of Problem (5.3) for a given $u \in [0,1]$. By definition of Problem (5.3), $\bar{z}$ violates the surrogate constraint. Thus $\bar{z}$ also violates one of the two constraints of $2O2DKP$. $\bar{z}$ is feasible for Problem (5.3) with $u = 0$ or $u = 1$, i.e. for Problem (5.1) based respectively on the second or the first constraint. As a consequence, the objective value of $\bar{z}$ is lower or equal than the one of z^0 or z^1, for Problem (5.3).

Example 15 gives an example of the solving of Problem (5.3) for $u \in]0,1[$.

Example 15. *Let us consider the $2DKP$ instance presented in Example 14, $x^{LP} = (0.26, 1, 1, 0.55, 0)$*

$z^0 = (1,1,0,1,0)$ is the optimal solution of Problem (5.3) with $u = 0$, its objective value is 0.74; $z^1 = (0,1,1,1,0)$ the optimal solution with $u = 1$ and its objective value is 0.2 (see Example 14).

For $u = 0.9$, Problem (5.3) is:

$$\begin{array}{lrcl} \max & 0.74\, z_1 + 0\, z_2 + 0.2\, z_3 + 0\, z_4 + 1\, z_5 & & \\ s.t. & 4\, z_1 + 2\, z_2 + 8.3\, z_3 + 3.7\, z_4 + 8.8\, z_5 & \geq & 14.4 \\ & z_j \in \{0,1\},\ j = 1 \ldots 5 & & \end{array}$$

Its optimal solution is $(1,1,1,1,0)$ *and its objective value is* 0.94. *The resulting cover inequality (*$x_1+x_2+x_3+x_4 \leq 3$*) is violated by* x^{LP}. *However, the cover* $\{1,2,3,4\}$ *obtained with* $u = 0.9$ *is included both in* $\{1,2,4\}$ *(obtained with* $u = 0$*) and* $\{2,3,4\}$ *(obtained with* $u = 1$*).*

Generally the covers found for $u \in]0,1[$ are included in the covers found for $u = 0$ or $u = 1$ (as in Example 15), i.e. they are less tight and redundant than the ones found using $u = 0$ or $u = 1$.

In some rare cases using a multiplier $u \in]0,1[$ can lead to a cutting cover inequality which is not included in the one obtained for $u = 0$ and $u = 1$. Finding such inequalities would be interesting, but the computational time required to find those rare inequalities may, most likely, compromise the tradeoff between computational time and quality. In this study we have chosen to analyze only the two constraints independently.

5.4.2 Generation of cover inequalities for $2O2DKP$

The valid inequalities generated in our branch-and-cut are based on the principles explained in Section 5.3.

Since Problems (5.1) and (5.2) were originally defined on the LP relaxation of a single-objective binary optimization problems, we have to adapt them to $2O2DKP$. Since the problem we deal with is bi-objective, then the LP relaxation does not give an optimal solution, but a set of efficient solutions. Problem (5.2) can thus be solved for any one of the efficient solutions of the LP relaxation, which are generally in infinite number. In particular, it can be solved for every extreme efficient solutions which are sufficient to define the upper bound set. In this section, we present a branch-and-cut method for $2O2DKP$. The cover inequalities introduced at each node are based on the extreme supported efficient solutions of the LP relaxation, by solving Problem (5.2), considering successively the two constraints.

We can remark that a cover inequality generated for a node can easily be adapted and used in its child nodes (since only a few variables have been fixed). The adaptation of a valid inequality is the following:

- for every variable in the valid inequality which has been fixed to 1, the right side of the inequality is reduced by one and this variable is deleted from the inequality;
- for every variable in the valid inequality which has been fixed to 0, the variable is deleted from the inequality, without changing the right side of the inequality. If the valid inequality is redundant (i.e. the sum of the coefficient on the right side is lower or equal to the left side), then the valid inequality can be deleted.

Example 16 illustrates how valid inequalities are adapted from the parent to the child node.

Example 16. *We consider the valid inequality* $x_2 + x_3 + x_4 + x_5 \leq 2$ *and suppose that the variable* x_3 *is fixed to 1 and next* x_5 *to 0. Since* x_3 *is fixed to 1, the valid inequality becomes* $x_2 + x_4 + x_5 \leq 1$. *Since* x_5 *is fixed to 0, the valid inequality becomes* $x_2 + x_4 \leq 1$. *This valid inequality is not redundant. It is the valid inequality inherited by the child node.*

Based on this observation, at a given node of the branch-and-cut method, we can use the valid inequalities generated for the parent node and even the one generated for all its ancestor nodes. Those inherited inequalities make it possible to have a tighter upper bound set in the child node.

One way to tighten the upper bound set at each node could be to:

Step (I) compute the LP relaxation of $2O2DKP$, to which we add the valid inequalities adapted from the ancestor nodes;

Step (II) generate cover inequalities based on every extreme efficient solutions of problem considered during Step (I), for both constraints;

Step (III) recompute the LP relaxation (with the cover inequalities generated during Step (II) and those of Step (I)).

The upper bound set obtained during Step (III) is tighter than the one obtained during Step (I), since it consider more cover inequalities. The upper bound set obtained by Step (III) is also defined by extreme solutions, so Step (II) and (III) can be repeated several times, tightening the upper bound set every time.

Intuitively, at a given node when the number of valid inequalities increases (thanks to Problemn (5.2) or inherited from the ancestor nodes), the upper bound is tighter. Therefore, performing several times Steps (II) and (III) leads to tighter upper bound sets.

First experiments showed that even if this process (Steps (I), (II) and (III)) is done once, the tradeoff between the computational time and the quality of the upper bound set is not interesting. This process leads to a reduction of the search-tree by 10%, however since a second continuous bi-objective problem has to be solved at each node, the computational time of the whole solution method is almost doubled (the computation of the upper bound set is, by far, the most time consuming component of the method). Since the valid inequalities can be inherited to the child nodes, the upper bound set obtained during Step (I) is tighter than if no valid inequality were used. Therefore, Step (III) can be omitted. The valid inequalities generated during Step (II) allow to tighten the upper bound set of the child nodes.

Moreover during the first experiments, we noticed that when the number of valid inequalities increases, the cost of adapting those valid inequalities and the cost of the computation of the upper bound set increase and the solution method is more time consuming, even if the search-trees are smaller. So we have to regulate the number of valid inequalities at each node. This question is discussed in Section 5.4.3

5.4.3 Strategies related to cover inequalities

To regulate the number of valid inequalities considered at each node, we can play on two levels of the algorithm: when the valid inequalities are generated or when they are adapted from a parent to a child node.

Let us first consider the generation of the cover inequalities. We recall that two cover inequalities (one for each constraint of the $2O2DKP$) can be generated for every extreme efficient solution of the LP relaxation. However, it is not necessary to analyze each one of those solutions. We can choose to generate cover inequalities only on a subset of those solutions, thus less cover inequalities are generated. This raises two questions: How many solutions should be analyzed? Which solutions should we analyze? In this study we will analyze different settings, answering those two questions.

Concerning the first question, we can easily set a maximum number of analyzed solutions at each node.

Concerning the choice of the solutions to analyze, we elaborated four strategies. Ideally, we would like to generate the "best" cover inequalities. However, this notion is vague and requires a characterization of the quality of a cover inequality. Thus we employ heuristic approaches, detailed here:

found The solutions are analyzed in the order they were found in the dichotomic method computing the LP relaxation. The lexicographic optimal solutions, which are more likely to be very different, are analyzed first.

distributed It analyses well distributed solutions of the upper bound set. Let us consider the upper bound set is composed of b extreme solutions and s solutions have to be analyzed ($s \leq b$). Then the solutions analyzed are the one at the positions $\left\lfloor \frac{b}{s+1} \right\rfloor, \left\lfloor \frac{2\,b}{s+1} \right\rfloor, \ldots, \left\lfloor \frac{s\,b}{s+1} \right\rfloor$, in the extreme solutions of the LP relaxation, in the natural order. The idea behind this heuristic is that the cover inequalities generated for a given solution are likely to cut also the adjacent extreme efficient solutions for the relaxed problem.

max0 It orders the solutions according to the number of variables with a value 0, and chooses the solutions to analyze in decreasing order of this measure. Based on Problem (5.1), the variables with a value of 0 do not appear in the generated cover inequality. Thus when the number of variables at 0 increases, it is the more likely to obtain small cover inequalities.

min0 It is the symmetric of *max0*, the solutions are considered in increasing order of the number of variables at 0. On the contrary to *max0*, using this strategy it is more likely to obtain bigger covers, which are more likely to be non-redundant for a large number of descendant node.

The second axis to regulate the number of cover inequalities is to select the valid inequalities to inherit to the child node. As for the generation of the cover inequalities, it is difficult to determine which one of the cover inequalities will be the most useful in the branch-and-cut method. Thus we propose four heuristic approaches:

smallestRh It selects the valid inequalities with the smallest right side. This strategy aims to select only one cover inequality. However, if several inequalities have this smallest right side, then they are all adapted to the child node.

biggestRh It is the symmetric of *smallestRh*, it selects the valid inequalities with the biggest right side. Similarly to the previous strategy, if several cover inequalities have the same value, they are all adapted.

ratio It selects the valid inequalities with the smallest ratio right side over number of variables of the valid inequality. Several inequalities can be adapted if they all have the smallest ratio.

active It adapts all the cover inequalities, except the ones that were not "active" during the computation of the upper bound set. A cover inequality is considered as active if the constraint derived from it is tight in at least one extreme efficient solution of the relaxation.

The branching strategies can also be based on the cover inequalities. We consider three cover-based branching strategies. The cover based branching strategies only focuses on the cover inequalities that will be inherited by the child node. The first one is called *cover* and it fixes the variable which appears the most often in those cover inequalities. The aim is to enforce the valid inequalities at the child nodes. The two last ones mix the two separation strategies *cover* and *nbFrac*. The separation called *coverNbFrac* is mainly based on the strategy *cover* and the equalities are discriminated thanks to *nbFrac*. The second strategy, called *nbFracCover*, is mainly based on the *nbFrac* and the equalities are broken by *cover*.

5.5 Experimental results

In this section, we evaluate the impact of the different variants of the branch-and-cut method presented in the previous section (number of solution analyzed, choice of the solutions to analyze,

choice of the cover inherited, branching strategy). We also analyze the impact of extending the cover inequalities, i.e. adding items, which have a larger weight, to the cover.

All variants of the branch-and-cut method are compared to the branch-and-bound method using the path relinking initialization, the LP based upper bound set and the *nbFrac* branching strategy. Since the instances with 28 variables have a low computational cost, the evaluation of those variants is done on the instances of the benchmark with 100 or 105 variables, in order to highlight the differences between the variants. The variants are compared regarding two aspects: the size of the search-tree and the computational time. The measures used for the comparison are again $\mathcal{IRS}$ for the first aspect and $\mathcal{IRT}$ for the second.

In first experiments, which we do not show here, we observed that increasing the number of analyzed solutions generally reduces the size of the search-tree but also increases the computational time. In more than 70% of the cases, the variants analyzing only one solution lead to the best computational time in average. In order to reduce the number of variants to analyze in this section we consider only variants considering one solution for Sections 5.5.1 and 5.5.2. However the influence of the number of analyzed solutions to generate valid inequalities is studied more in detail in Section 5.5.3.

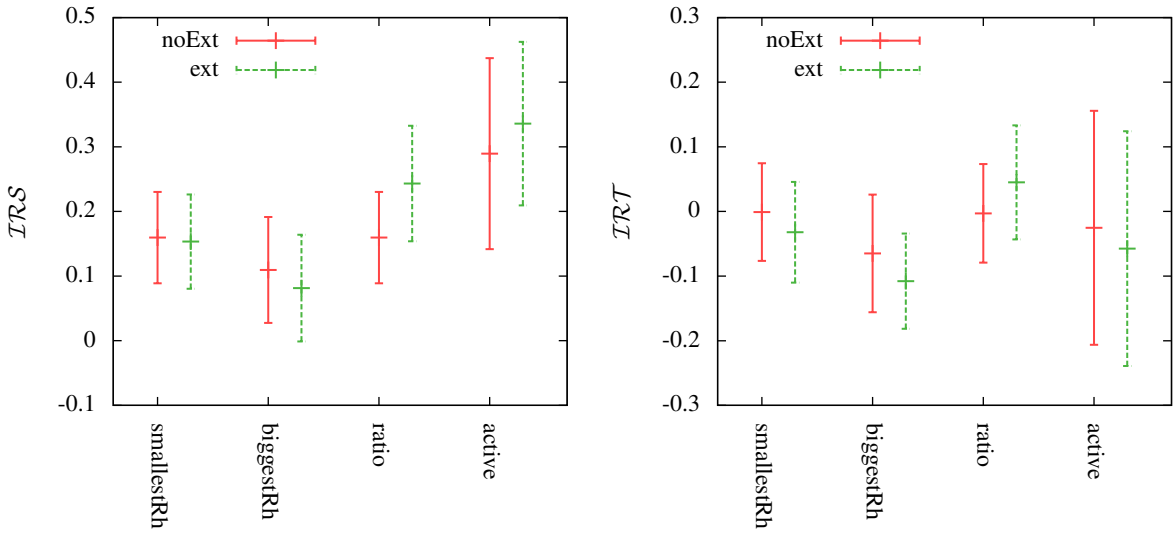
5.5.1 Impact of the extension of the cover inequalities

To identify the most efficient way to generate the valid inequalities used during the branch-and-cut component, we first consider the benefit to extend the generated cover inequalities. We recall that a cover inequality is extended by adding in the left side of the inequality every item whose weight is larger than the ones in the cover, without changing the right side of the inequality. Therefore, the obtained cover inequality is tighter. We expect that the impact of the extension of the cover inequalities depends on the covers inherited by the child node. For example the strategy *ratio* is more likely to take advantage of the extension of cover inequality than the strategy *biggestRh*.

Figure 5.7 compares the variants using the extended cover inequalities (named *ext*) and the ones using non-extended cover inequalities (named *noExt*). The results are separated according to the heuristic of choice of the covers inherited by the child node. The presented results are the average and standard deviation of $\mathcal{IRS}$ and $\mathcal{IRT}$, over all instances and over the different possible variants of the choice of the solutions to analyze. For all variants of the branch-and-cut method, the upper bound set the branching strategy is *nbFrac* and the path relinking initialization is used.

Figure 5.7a shows that the extension of covers inequalities leads to a reduction on average of the size of the search-tree in the variants *ratio* and *active*. The performance obtained by extending or not the cover inequalities for the variants *smallestRh* are equivalent. However, using extended cover inequalities does not lead to better results for the variant *biggestRh*. This last result can be explained by the fact that the branching strategy is dynamic (*nbFrac*). Indeed the introduced inequalities change the obtained upper bound set and thus also influence this separation strategy. Therefore, the size of the search-tree may increase when using extended cover inequalities.

As expected, Figure 5.7b shows that the extension of cover inequalities does not have the same impact on the computational time depending on the strategy of choice of the inherited covers inequalities. The only strategy for which the extension allows to reduce the computational time is *ratio*.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 5.7: Impact of the extension of cover inequalities

5.5.2 Choice of the solutions to analyze and choice of the inherited cover inequalities

Figure 5.8 compares the performances obtained for different heuristics of choice of the solution to analyze and for different heuristics of choice of the cover inequalities inherited by the child node. According to the results presented in Section 5.5.1, we consider extended cover inequalities when dealing with the heuristic *ratio* and non-extended ones for *smallestRh*, *biggestRh* and *active*. For every variant tested here, the branching strategy is *nbFrac* and the path relinking initialization is used.

Both Figures 5.8a and 5.8b show that the heuristics *max0* performs better or equivalently than the other heuristics, independently of the considered heuristic of choice of the inherited cover.

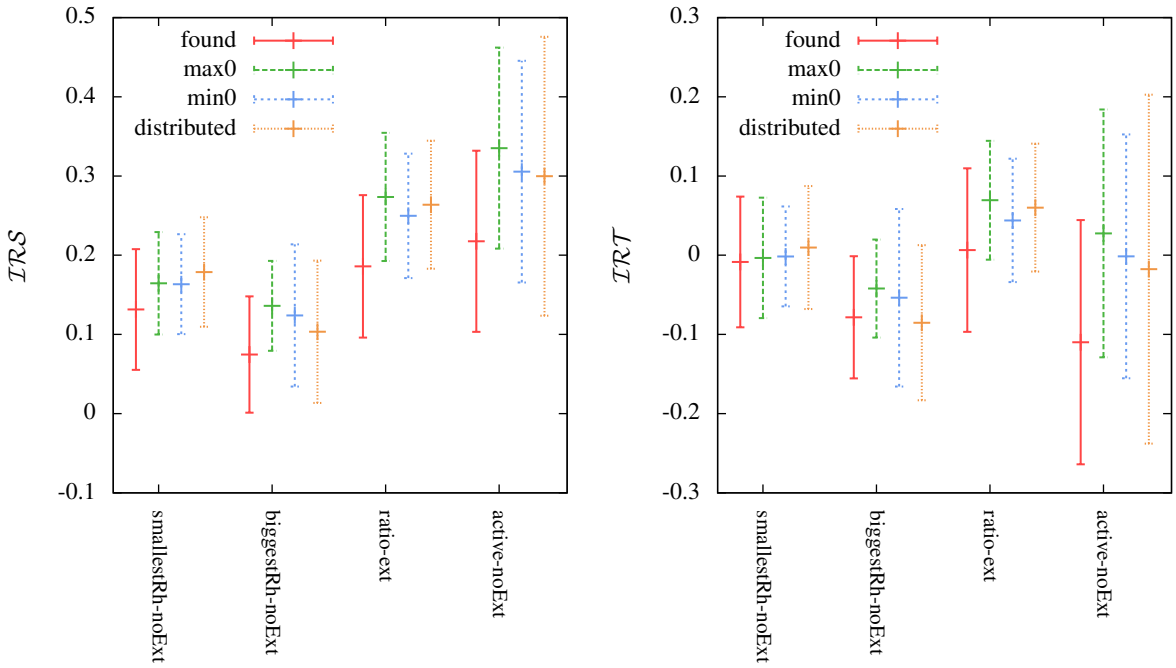
According to the size of the search-tree, the best heuristic of choice of the inherited cover inequalities is *active-noExt*, since it leads to an higher $\mathcal{IRS}$ than the other ones, regardless of the heuristic used to choose the solution to analyze. However, this heuristic is expensive and it does not give the best results regarding the computational time.

Figure 5.8b shows that the heuristic *ratio-ext* has the higher average $\mathcal{IRT}$. The overall best variant of branch-and-cut method, regarding the computational time, considers the heuristics *ratio-ext* and *max0*.

5.5.3 Number of solutions analyzed

In this section, we are interested in understanding the impact of the number of analyzed solutions, at each node, to generate the cover inequalities. Figure 5.9 compares different settings for this parameter, in a branch-and-cut method using the combination of strategies *ratio*, *ext* and *max0* to generate and adapt the cover inequalities, the *nbFrac* branding strategy and the path relinking initialization.

Figure 5.9a shows that the number of analyzed solutions does not impact significantly the size



(a) According to the size of the search-tree

(b) According to the computational time

Figure 5.8: Comparison of the heuristics of choice of the covers inherited and the heuristics of choice of the solutions analyzed

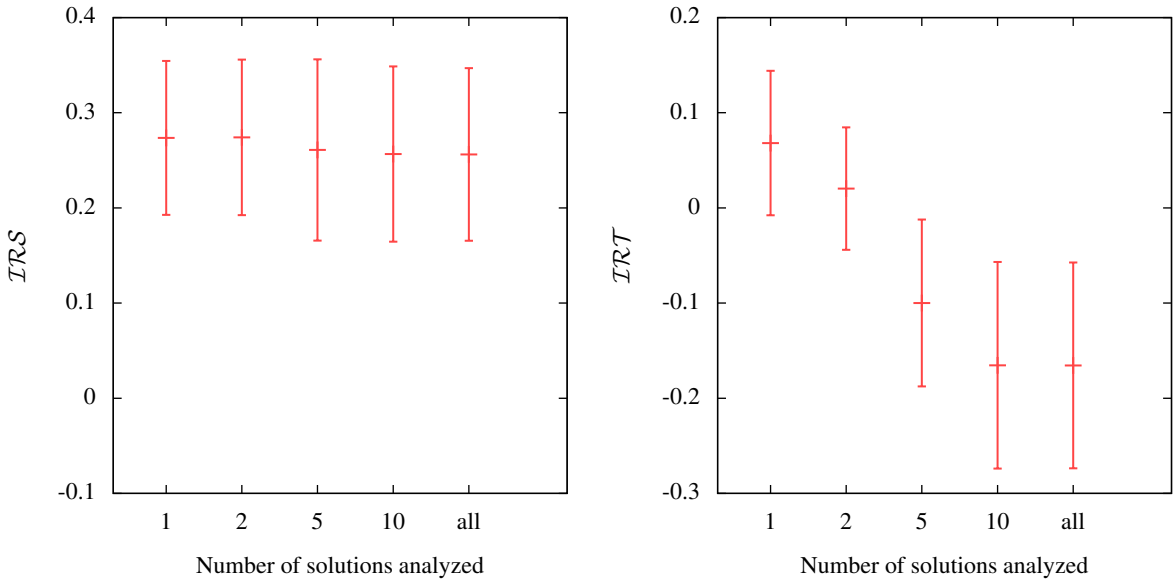
of the search-tree. Analyzing a single solution at each node allows an average reduction of 27% of the size of the search-tree and analyzing more solutions does not lead to a larger reduction. Indeed, the cover inequalities generated for different solutions may not allow to cut more nodes. Moreover, the branching strategy (*nbFrac*) is dynamic and can thus have a negative influence on the size of the search-tree.

Figure 5.9b shows that analyzing only one solution at each node of the branch-and-bound tree leads to the best performance. When more solutions are analyzed the computational time spent on the generation of cover inequalities is not counterbalanced by the reduction of the search-tree.

5.5.4 Branching strategies

Figure 5.10 presents the results obtained for the five branching strategies, presented in the previous section. The applied method is the branch-and-cut method, analyzing one solution at each node and using the combination of strategies *ratio*, *ext* and *max0*. The path relinking initialization is used.

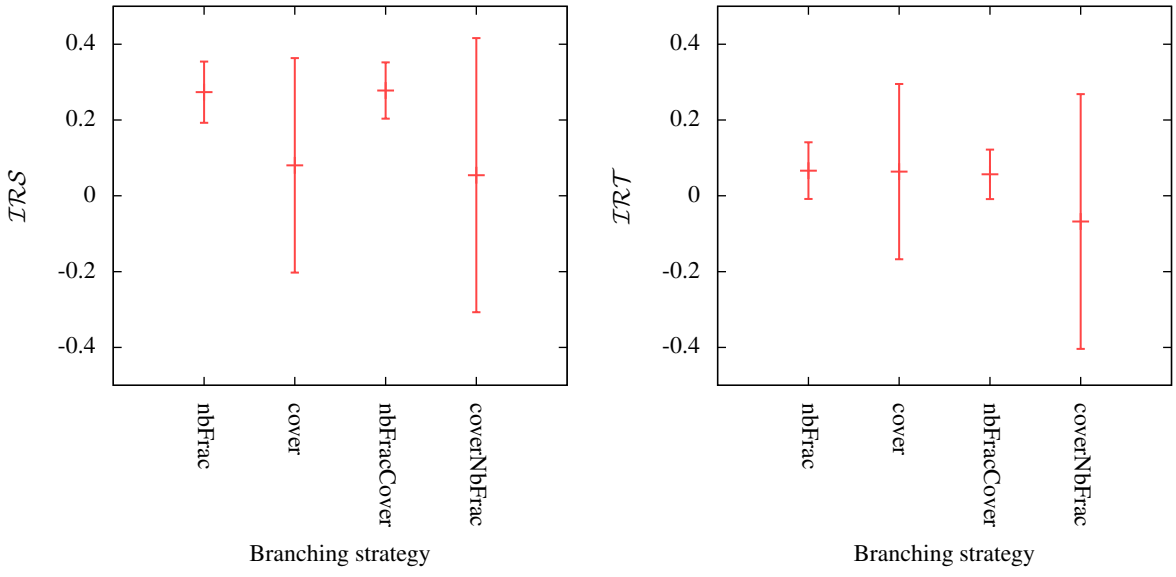
Figure 5.10a shows that the branching strategies *nbFrac* and *nbFracCover*, which are based mainly on the number of solutions for which the variables are fractional, lead to better results than the two others, with respect to the average size of the search-tree. The difference between the branching strategies *nbFrac* and *nbFracCover*, and the strategies *cover* and *coverNbFrac* is less important regarding the execution time. The branching strategies *nbFrac* and *nbFracCover* give equivalent performances regarding the size of the search-tree and the computational time. *nbFrac* gives slightly lower computational time on average.



(a) According to the size of the search-tree

(b) According to the computational time

Figure 5.9: Impact of the number of solutions analyzed for the combination of strategies *max0, ratio, ext*



(a) According to the size of the search-tree

(b) According to the computational time

Figure 5.10: Impact of the number of solutions analyzed for the strategy *max0, ratio, ext*

5.5.5 Comparison to the ε-constraint method

Table 5.6 compares the branch-and-bound and the branch-and-cut methods, embedded in a two phase method, along with an ε-constraint method. For the branch-and-bound and branch-and-cut method, the path relinking initialization is applied, the upper bound set is based on the

instance indicator	nb efficient	epsilon constraint time (s)	Branch-and-bound based nb nodes	Branch-and-bound based time (s)	Branch-and-cut based nb nodes	Branch-and-cut based time (s)
A1	24	**1,462**	16880	3,052	6482	*1,936*
A2	26	**1,343**	9535	2,140	6721	*2,011*
A3	19	**1,691**	19177	3,699	11263	*3,334*
A4	58	1,197	4977	***1,179***	4307	1,345
D1	35	**0,426**	3740	0,928	2642	*0,895*
D2	63	1,340	3616	***0,883***	3360	1,007
D3	54	**0,693**	3853	*0,875*	3487	1,009
D4	24	**0,440**	4517	*0,921*	4107	1,074
kp28	85	0,820	3189	***0,758***	2753	0,805
kp28-2	35	0,281	718	***0,259***	616	0,281
kp28W-Perm	41	**0,268**	1237	0,426	975	*0,417*
kp28W-ZTL	6	0,066	123	0,063	95	***0,062***
kp28W	29	**0,218**	1108	*0,359*	1080	0,400
kp28c1W-c2ZTL	21	**0,196**	1235	*0,305*	1067	0,347
kp28cW-WZTL	132	**2,412**	19382	4,283	13284	*4,088*
ZTL28	18	**0,193**	1599	*0,435*	1213	0,452
rnd1-1800	239	**4,931**	66138	23,253	52580	*22,823*
rnd1-3000	73	**0,906**	6725	*2,442*	5419	2,611
tube1-1800	252	**12,789**	233222	84,783	139382	*64,877*
tube1-3000	100	**2,270**	13906	4,860	9222	*4,777*
tube1-asyn	394	**6,012**	66421	21,257	41851	*19,933*
tube2-1800	282	**18,036**	235477	79,162	161021	*71,807*
collage-tube	330	**12,325**	60145	*24,645*	50797	25,903
ZTL100	121	**6,124**	154884	54,799	118662	*53,694*
ZTL105	152	**11,237**	263214	108,516	197896	*103,439*
ZTL150-f250	249	**39,321**	990693	408,476	791737	*408,357*
ZTL150-f500	277	**44,790**	1337769	592,297	1095073	*584,909*
ZTL200-f250	377	**105,084**	3819631	1942,580	2959873	*1803,420*
ZTL200-f500	337	**75,907**	2710440	*1457,240*	2280104	1553,450
ZTL250	568	**239,621**	-	-	-	-

Table 5.6: Comparison of the ε-constraint method, the branch-and-bound based method and the branch-and-cut based method (*nbFrac, max0, ratio, ext, 1 sol*)

LP relaxation and the branching strategy is *nbFrac*. The strategies applied for the branch-and-cut method are *max0*, *ratio*, *ext* and one solution is analyzed at each node to generate the cover inequalities.

In Table 5.6, the column *nb efficient* indicates the number of efficient solutions for each instance. The columns *time (s)* indicates the computational time, in seconds, and *nb nodes* indicate the size of the search-trees obtained for the branch-and-bound and branch-and-cut methods. The table is vertically divided according to the size of the instances (28, 100 or 105, 150, 200 and 250 variables). "-" indicates the instances whose computational time exceed the time limit of one hour. For each instance, the best computational time among the three methods is indicated in bold and the best computational time among the branch-and-bound and the branch-and-cut methods is indicated in italic green.

Table 5.6 shows that the computational time obtained for the branch-and-bound based and branch-and-cut based methods are equivalent for the instance of 28 variables. However, the difference between these two methods seems to enlarge when the size of the instances increases. The size of the search-tree is always lower for the branch-and-cut method than for the branch-and-bound method. Over the 30 instances, the branch-and-cut method has a lower computational time than the branch-and-bound method on 17 instances and for 10 over the 13 instances with 100 or more variables.

Generally the ε-constraint method offers smaller computational time than the two others. If the difference is small for the instances with 28 variables, it is more important for bigger instances. This difference might be the result of the implementation of the LP relaxation solving. Indeed, as explained at the beginning of the chapter, in order to avoid numerical imprecision, the LP relaxation is solved by a dichotomic method. However it would be less time consuming to compute it by a parametric simplex method. Since the LP relaxation is the main cost of the solution method, changing its implementation should allow the branch-and-bound and branch-and-cut methods to be competitive with the ε-constraint method.

5.6 Conclusion

In this chapter, we implemented a two phase method to solve $2O2DKP$, whose second phase is either a branch-and-bound method or a branch-and-cut method. In the first part, we compare three relaxations to compute the upper bound set in the branch-and-bound method. As expected, the convex relaxation leads to the smallest search-trees. However, since the single-objective version of $2O2DKP$ is not solvable in polynomial or pseudo-polynomial time, solving the convex relaxation is expensive. Therefore, using this relaxation leads to high computational time. The LP relaxation gives the smallest computational times, however the size of the obtained search-trees are largely bigger than those obtained for the convex relaxation.

In order to reduce the size of the search-tree obtained when using the LP relaxation, cover inequalities are introduced along the solving process. Those inequalities are generated based on the efficient solutions of the LP relaxation, adapting the method presented in (Crowder et al., 1983). The quality of the LP relaxation increases with the number of introduced cover inequalities. However the computational cost also increases. It was then necessary to find a tradeoff between the quality of the upper bound set and its computational cost. Multiple variants of the branch-and-cut method have been compared in order to find this tradeoff. These variants considered several implementations of the different components of the branch-and-cut method: number of analyzed solutions to generate the cover, choice of the analyzed solutions, use of extended covers, choice of the cover inequalities inherited by the child node. Experiments show that introducing cuts during the solving process allows to reduce the computational time in most of the instances (56%) and in particular for big instances (77% of the instances with more than 100 variables). However the solving method remains slower than the ε-constraint method, in particular because of the implementation of the solution method for the LP relaxation.

This chapter also highlights that using a dynamic branching strategy based on the analysis of the efficient solutions of the LP relaxation makes it possible to improve the solving process. We have also presented an initialization method, executing path relinking operators between supported efficient solutions, in order to initialize the solution method in each triangle with good feasible solutions. This initialization has been experimentally validated and compared to an ideal initialization (initializing with all efficient solutions).

In order to improve the practical efficiency of the branch-and-cut method, we want to bring several improvements to the implementation. The first one concerns the solution method for the LP relaxation. Instead of using a dichotomic method, we will implement a parametric simplex method, which will reduce the cost of the solution method. Moreover, we are looking for using informations from the computation of the upper bound set of the parent node to speed-up the computation of the upper bound set of the child node. On another hand, several additional instances could be generated in order to assess more accurately the performances of the branch-and-cut based method. Indeed, only a few $2O2DKP$ instances of more than 105 variables are available.

Finally, the branch-and-cut method could be generalized to more complex $pOmDKP$, with more constraints for example, or to other MOCO problems.

Conclusions and perspectives

This thesis aims to elaborate an efficient solution method for multi-objective combinatorial optimization problems. The contributions of this work are structured in two axes: the analysis of different components in order to propose a new branch-and-bound method, and the elaboration of a new branch-and-cut method for a bi-objective combinatorial optimization problem. The multi-objective knapsack problem is used as support problem of these researches.

The first two chapters of this thesis present the state of the art on multi-objective combinatorial optimization problems and focus on branch-and-bound methods and knapsack problems. We identified three aspects that could be improved.

Branching strategies Many strategies have been proposed over the years to order the variables during the solving process. Several works present a comparison of different strategies, for example in Bazgan et al. (2009a), Jorge (2010), Delort (2011) and Florios et al. (2010). However, these studies do not pinpoint a best strategy over all the instances. Indeed, since most of the proposed strategies are static, they usually present good performances for some instances, but not necessarily for other ones. Moreover, these studies focus on a restricted number of strategies and no exhaustive comparison of all branching strategies have been proposed.

Surrogate relaxation The use of surrogate relaxation becomes more difficult when considering several objective functions than for the single-objective context. Indeed, in the single-objective context and for two dimensions, an algorithm allowing to compute the dual surrogate (the tightest surrogate relaxation) in reasonable time has been proposed by Fréville and Plateau (1993). In multi-objective contexts, the surrogate relaxation is generally used considering several surrogate multipliers (Gandibleux and Perederieieva (2011)), but no guarantee is given regarding the quality obtained.

Branch-and-cut methods If they have presented interesting results for single-objective problems, very few works have considered the generalization of branch-and-cut methods to multi-objective problems. Jozefowiez et al. (2012) have proposed to execute several single-objective branch-and-cut methods to solve a multi-objective problem.

These observations have defined the three main parts of this thesis.

Chapter 3 relates to the branching strategies in branch-and-bound methods. In this chapter, we first compared different static strategies of the literature and observed that none of them gives the best performance for all instances. Thus, we investigated if using different strategies, during a single execution of branch-and-bound, could improve the performance of the solution method. We proposed a method (the oracle method) that tests all strategies at each separation, for the next step only, and chooses the best one according to a quality measure. This method appears to be able to significantly reduce the size of the resulting search-trees. However, its computational cost is significantly higher than the one obtained by a static strategy, even when using a restricted set of branching strategies. Then, reinforcement learning methods are used to select dynamically

one branching strategy at each separation. Numerous dynamic strategies, defined using adaptive methods, have been experimentally assessed. The one offering the best results allows to reduce the computational time for 58% of the instances, compared to a static strategy.

This chapter focuses on the application problem: the uni-dimensional bi-objective knapsack problem. It would be interesting to generalize this work for other multi-objective combinatorial problems. On the other hand, even if this work has considered only the branch-and-bound method, it is directly applicable to other methods in which an order on the variables has to be defined (dynamic programming method for example).

Chapter 4 consists in defining and computing tight upper bound sets for the bi-objective bi-dimensional knapsack problem, based on the surrogate relaxation. Two upper bound sets are defined : the optimal surrogate upper bound set (OSUB) and the optimal convex surrogate upper bound set (OCSUB). These upper bound sets use several multipliers for the surrogate relaxation. The OSUB is the tightest possible upper bound set based on the surrogate relaxation. The OCSUB is the tightest upper bound set based on the convex relaxation of the surrogate relaxation. First, we defined an enumerative method to compute these upper bound sets. Then, the dominance relations between the surrogate relaxations are analyzed and lead to a more efficient method, considering a reduced set of interesting surrogate multipliers. The correctness of these two algorithms is proven and they are compared experimentally. A heuristic version of the latter algorithm allows us to obtain an approximation of the upper bound sets, in a more restricted time. This approximation is compared to another approximation method from Gandibleux and Perederieieva (2011) and appears to offer an interesting tradeoff between computational time and quality of the obtained upper bound set.

This chapter offers a theoretical basis on the surrogate relaxation for the bi-objective bi-dimensional knapsack problem. In future research, we would like to generalize this theory to knapsack problems with a larger number of objectives and/or constraints. Regarding the generalization to more than two objectives, the challenge comes from the computation of the convex relaxation of the surrogate relaxation. Several works can be used, e.g., Przybylski et al. (2010b), to solve this relaxation. However, the generalization to more than two dimensions is more challenging. Indeed the properties of the convex surrogate upper bound sets might not hold. Finally, OCSUB suffers from its computational cost when applied in a two phase method. So it would be interesting to consider more intensively the triangle investigated when computing the OCSUB in this context.

Chapter 5 aims to design a branch-and-cut method for the bi-objective bi-dimensional knapsack problem. Figure 5.11 presents the different components of the branch-and-cut method and their interaction.

We can note that by deleting the components *Generate cover inequalities* and *Inheritance of the cover inequalities by the child nodes*, we obtain the scheme of a branch-and-bound method.

In this chapter, different variants for the components of the branch-and-cut method, marked by a *, are tested. In particular, several relaxations are considered to compute the upper bound set on $\bar{Y}_N$ (the nondominated set of the subproblem). If the convex relaxation leads to small search-trees, the required computational time is too high. On the contrary, the LP relaxation gives small computational times, however the size of the obtained search-trees is considerably higher than for the convex relaxation. This leads us to introduce valid inequalities all along the solving process in order to improve it. The inequalities used are the extended cover inequalities. They are generated based on extreme efficient solutions of the LP relaxation. By doing so we generalize the principle presented by Crowder et al. (1983) for the single-objective context. When the number of considered cover inequalities increases, the quality of the upper bound set increases,

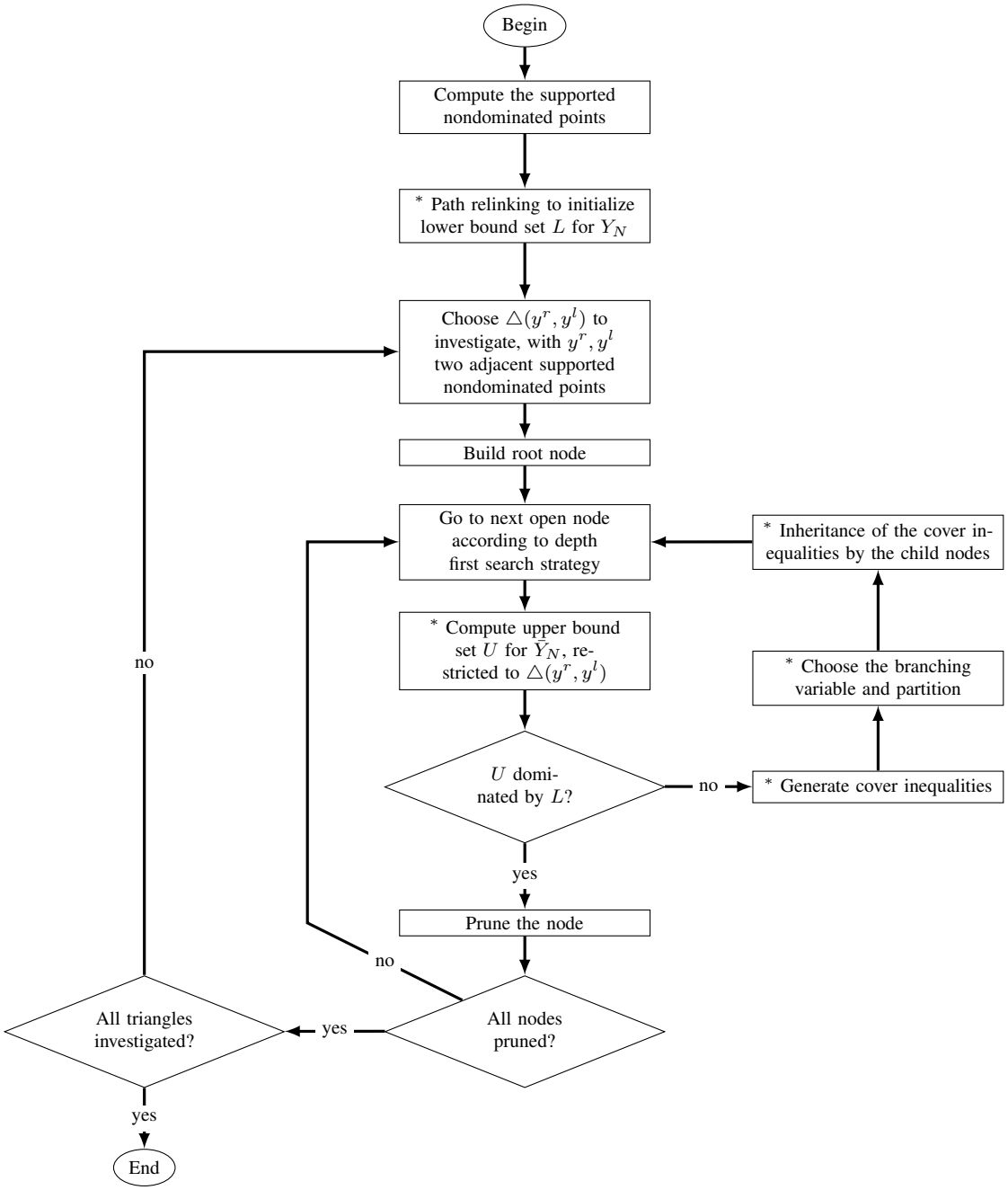


Figure 5.11: Main components of the branch-and-cut algorithm for $2O2DKP$

but the computational cost increases too. To achieve a good tradeoff between computational cost and quality of the upper bound set, several implementations have been defined and numerically tested. The strategies differing in these implementations are: the number of analyzed solutions to generate cover inequalities, the choice of these solutions, the choice of the cover inequalities adapted to the child nodes and the extension of the cover inequalities. The obtained branch-and-cut method allows to reduce the computational time, in particular for large instances, compared to the branch-and-bound method. However, this method is not competitive with an ε-constraint method. By using another method to solve the LP relaxation, executing a parametric simplex method, the branch-and-cut method should be competitive with the ε-constraint method. This branch-and-cut

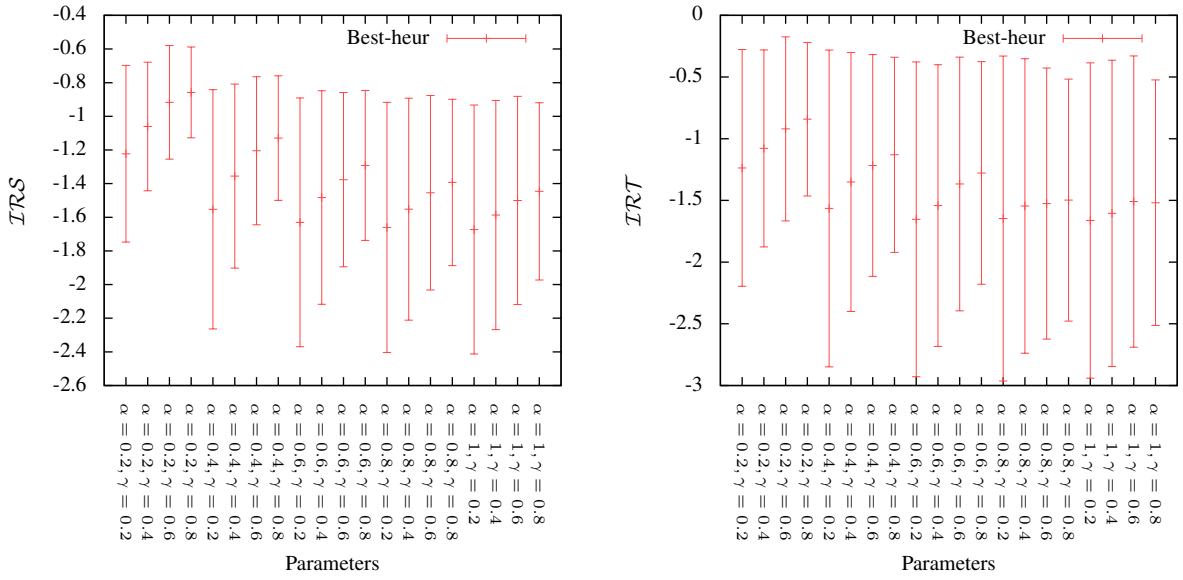
method could also be applied to other multi-objective combinatorial problems.

Figure 5.11 highlights the links between the different parts of this thesis. The different mechanisms can be assembled to design a solution method hopefully efficient in practice. Indeed, the basis given by Chapter 3 on the elaboration of a dynamic branching strategy could be adapted to the different branching strategies presented in Chapter 5 for the branch-and-cut method. An interesting perspective consists in designing an adaptive method mixing the strategies *nbFrac*, *sumFrac* and *cover* for example. Even if the surrogate-based upper bound sets is too expensive to be computed in every node of a branch-and-bound method, by the algorithms presented in this thesis, we could compute them punctually, in order to prune branches earlier in the branch-and-cut method. Rules defining when to compute this upper bound set need to be defined.

Adaptive methods as branching strategies

A.1 Dynamic multi-armed bandit

Figures A.1 and A.2 show the results obtained for different values of the parameters α and γ in the DMAB method.

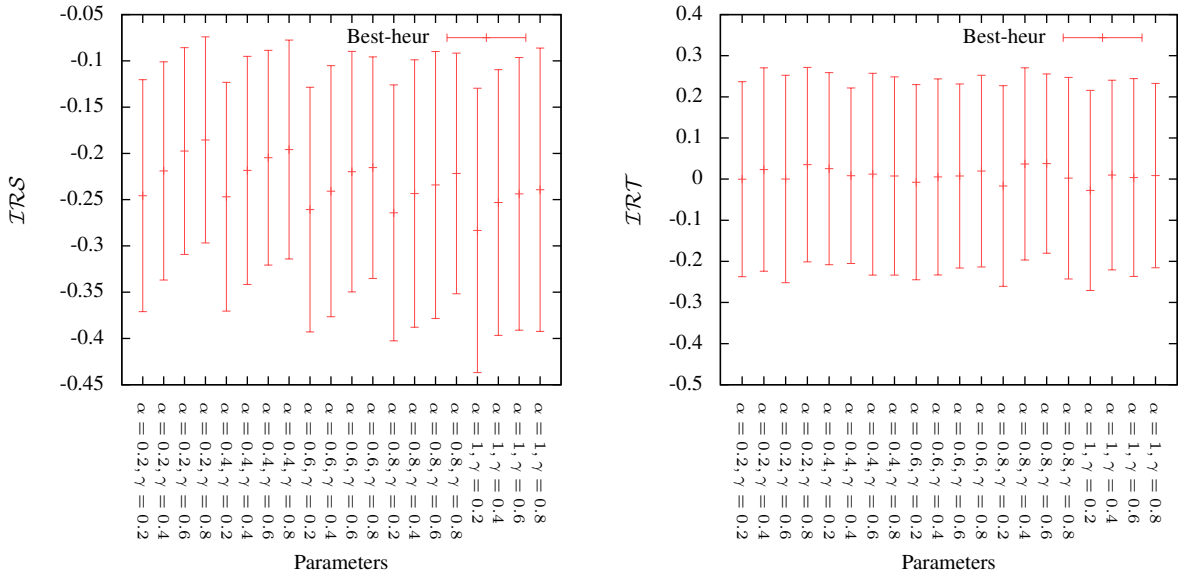


(a) According to the size of the search-tree

(b) According to the computational time

Figure A.1: Impact of the parameters on the solution method using the DMAB as branching strategy when no preprocessing treatment is applied.

We can observe that the average $\mathcal{IRS}$ and $\mathcal{IRT}$ increase when γ increases and also when α



(a) According to the size of the search-tree

(b) According to the computational time

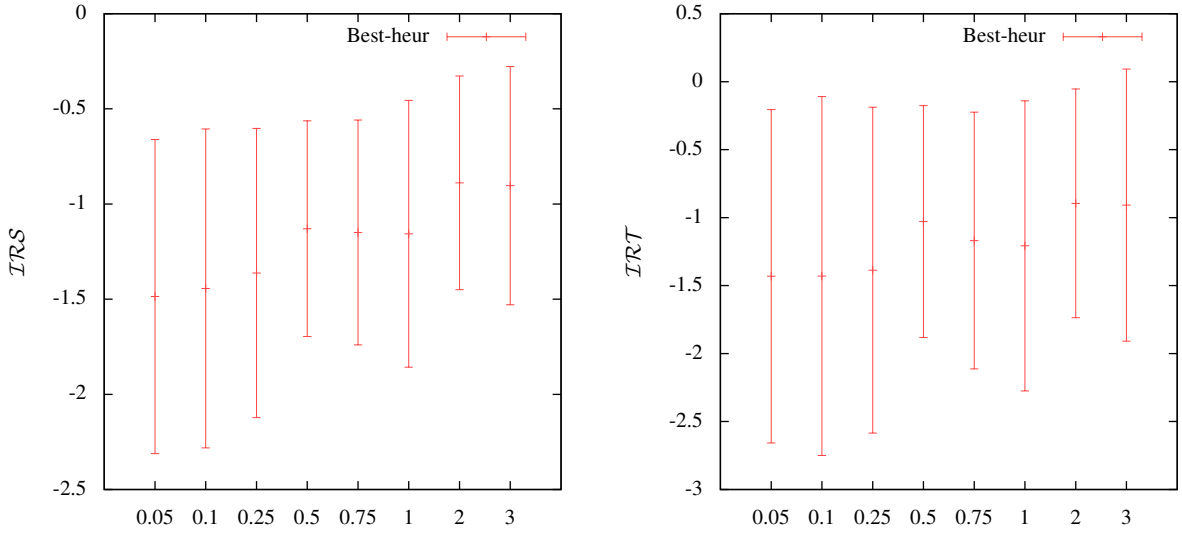
Figure A.2: Impact of the parameters on the solution method using the DMAB as branching strategy when the preprocessing treatments are applied.

decreases, whether the preprocessing treatments are used or not.

A.2 Upper confidence bound with a sliding window

The size of the window considered for the evaluation of the empirical quality of the branching heuristic is proportional to the number of variables in the instance. Indeed changes might occur more quickly when the instance considers only a few variables than when the instance considers a large number of variables. The size of the window is controlled by a parameter $s \in \mathbb{R}$, the size of the window is s times the number of variables in the instance. Figures A.3 and A.4 show the performances obtained for different values of the parameter s. We have fixed $\alpha = 0.2$ since it was the best value for UCB and DMAB.

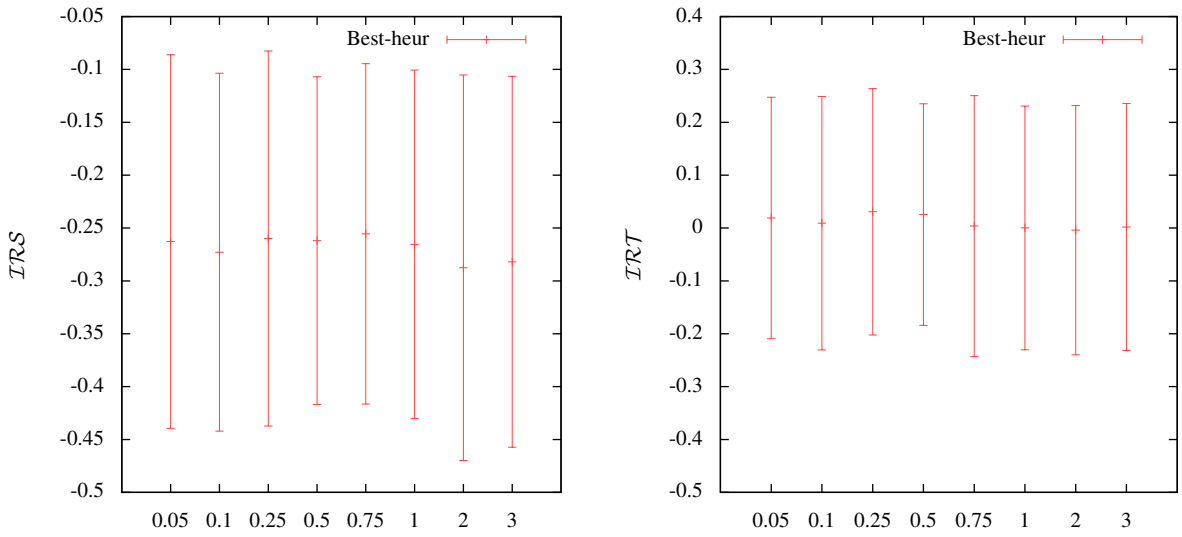
It seems that the better performances in average are given by $s = 0.75$. However, the difference of performance between the different values of the parameter is not very important.



(a) According to the size of the search-tree

(b) According to the computational time

Figure A.3: Impact of the size of the window on the solution method using the UCB with a sliding window as branching strategy when no preprocessing treatment is applied.



(a) According to the size of the search-tree

(b) According to the computational time

Figure A.4: Impact of the size of the window on the solution method using the UCB with a sliding window as branching strategy when the preprocessing treatments are applied.

B

OCSUB and OSUB

B.1 Computation of an upper bound on ϵ, in the algorithm computing the OCSUB or the OSUB

In all the algorithms presented in Chapter 4, ϵ is added to critical multipliers u such that $u + \epsilon \neq u$ and there does not exist a critical multiplier u' with $u < u' \leq u + \epsilon$ or $u + \epsilon \leq u' < u$. In these conditions we can find all critical multiplier associated to CSUB.

Our purpose is here to find $\epsilon > 0$ such that it is smaller than the smallest difference between two different critical multipliers.

To find a valid value for ϵ, we consider u and u' two different critical multipliers, such that the difference between u and u' is the smallest difference between two different critical multipliers. x is a solution associated to u and x' is a solution associated to u'. We have shown in Section 4.3.1 that $u = \dfrac{\omega_2 - w_2(x)}{w_1(x) - w_2(x) - \omega_1 + \omega_2}$. To simplify the notation we denote $p_i = w_i(x), q_i = w_i(x')$ for $i = 1, 2$. We have therefore

$$
\begin{aligned}
|u - u'| &= \left| \frac{\omega_2 - p_2}{p_1 - p_2 - \omega_1 + \omega_2} - \frac{\omega_2 - q_2}{q_1 - q_2 - \omega_1 + \omega_2} \right| \\
&= \frac{|(\omega_2 - p_2)(q_1 - q_2 - \omega_1 + \omega_2) - (\omega_2 - q_2)(p_1 - p_2 - \omega_1 + \omega_2)|}{|(p_1 - p_2 - \omega_1 + \omega_2)(q_1 - q_2 - \omega_1 + \omega_2)|}.
\end{aligned}
$$

If the numerator equals 0 then $u = u'$, and we make the assumption that both multipliers are different. Its value is then greater or equal to 1. Now we search the maximum value of the denominator, denoted $|D|$.

$$
\begin{aligned}
D =& (p_1 - p_2 - \omega_1 + \omega_2)(q_1 - q_2 - \omega_1 + \omega_2) \\
D =& p_1q_1 + p_1\omega_2 + p_2q_2 + p_2\omega_1 + \omega_1q_2 + \omega_1^2 + \omega_2q_1 + \omega_2^2 \\
& - (p_1q_2 + p_1\omega_1 + p_2q_1 + p_2\omega_2 + \omega_1q_1 + 2\omega_1\omega_2 + \omega_2q_2) \\
D \leq& \max(p_1q_1 + p_1\omega_2 + p_2q_2 + p_2\omega_1 + \omega_1q_2 + \omega_1^2 + \omega_2q_1 + \omega_2^2) \\
& - \min(p_1q_2 + p_1\omega_1 + p_2q_1 + p_2\omega_2 + \omega_1q_1 + 2\omega_1\omega_2 + \omega_2q_2)
\end{aligned}
$$

Since p_1 is the weight of the solution on the first dimension, we have $0 \leq p_1 \leq M_1$ with $M_1 = \sum_{j=0}^{n} w_{1j}$. Similarly $0 \leq q_1 \leq M_1$. Symmetrically $M_2 = \sum_{j=0}^{n} w_{2j}$, $0 \leq p_2 \leq M_2$ and $0 \leq q_2 \leq M_2$.

We can thus deduce that:

$$D \leq M_1^2 + 2M_1\omega_2 + M_2^2 + 2M_2\omega_1 + \omega_1^2 + \omega_2^2 - 2\omega_1\omega_2 = D_{max}$$

and similarly

$$D \geq \omega_1^2 + \omega_2^2 - 2\omega_1\omega_2 - 2M_1M_2 - 2M_1\omega_1 - 2M_2\omega_2 = D_{min}$$

Thus $|D| \leq \max(|D_{min}|, |D_{max}|)$ and $\epsilon < \dfrac{1}{\max(|D_{min}|, |D_{max}|)}$.

$\dfrac{1}{\max(|D_{min}|, |D_{max}|)}$ is an upper bound on ϵ.

B.2 Example of comparison of the OSUB and the OCSUB

Figure B.1 presents the OCSUB and the OSUB for the instance `A2`.

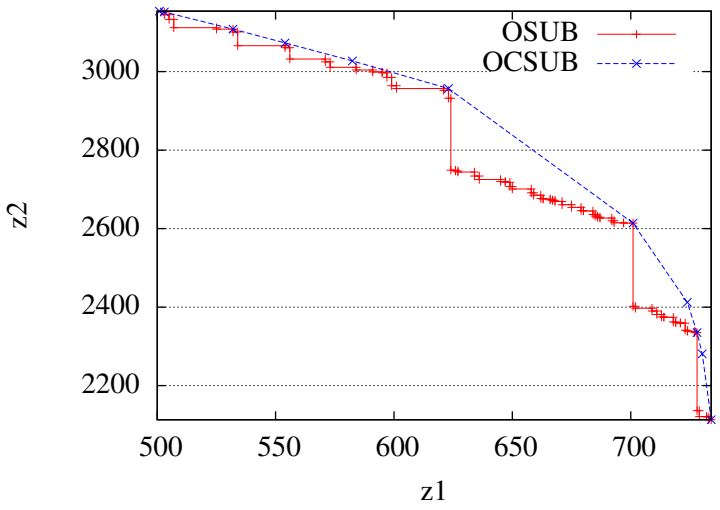
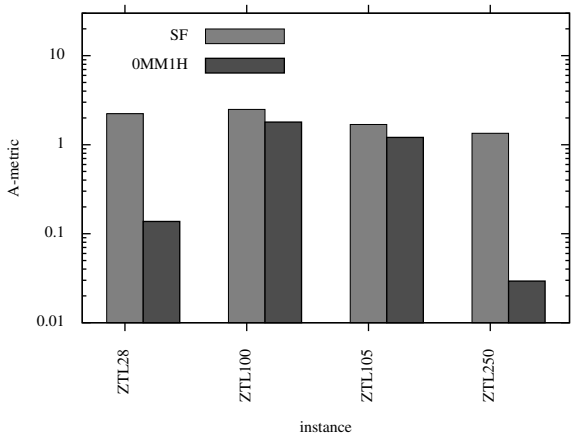


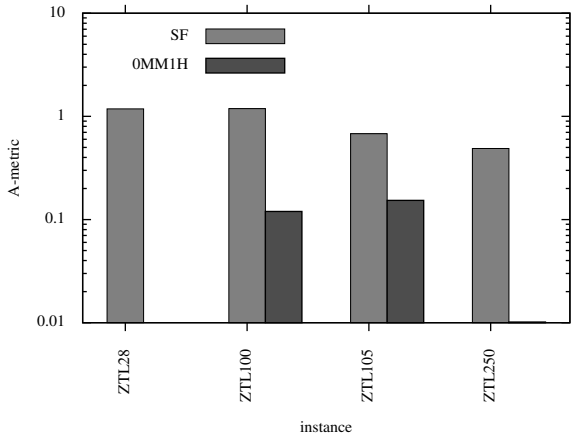
Figure B.1: Comparison of the OSUB and the OCSUB for the instance `A2`.

B.3 Exact and heuristic approaches for the OSUB

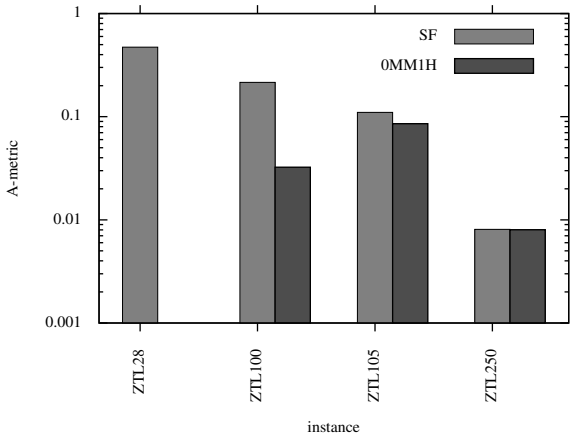
Figures B.2, B.3 and B.4 present the results obtained by the approximation method to compute the OSUB. The me sure used are similar to the comparison of the approximation methods for the OCSUB in Section 4.8.5.



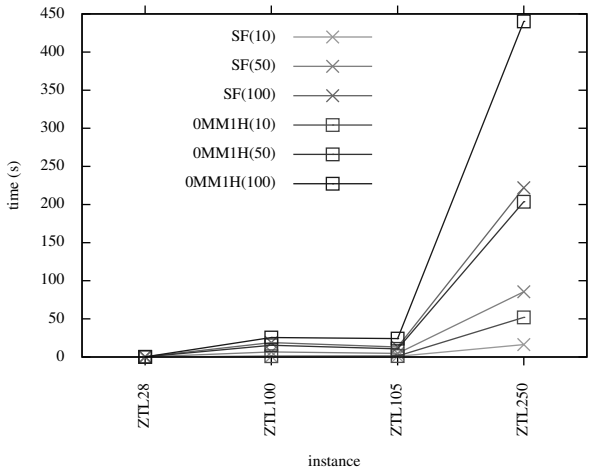
(a) Comparison of $\mathcal{A}$-metric for $h = 10$



(b) Comparison of $\mathcal{A}$-metric for $h = 50$

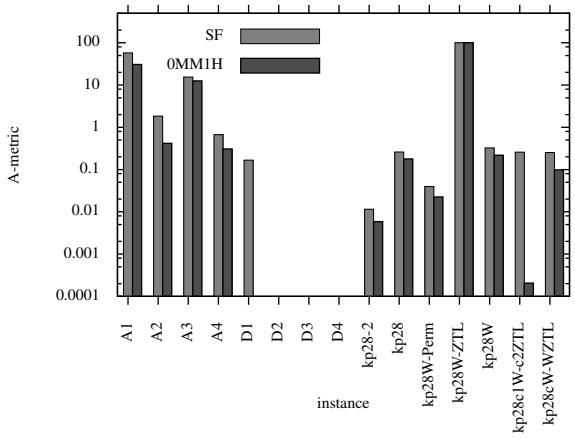


(c) Comparison of $\mathcal{A}$-metric for $h = 100$

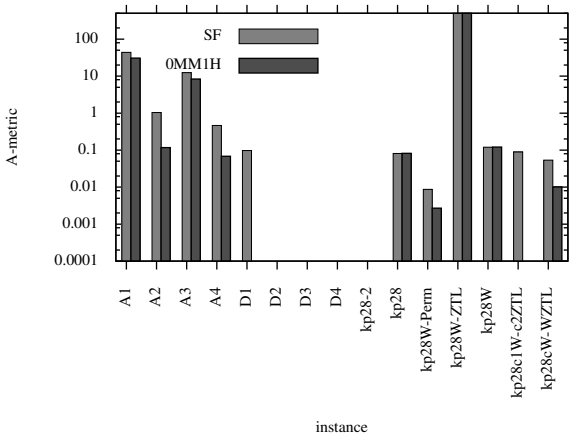


(d) Comparison of computational times

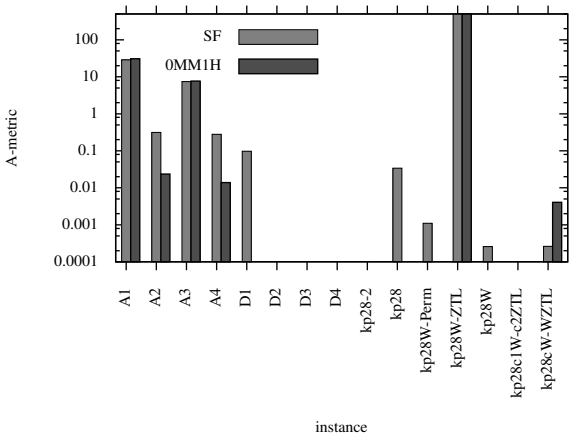
Figure B.2: Comparison of both heuristics *0MM1H + init* and *SurrogateFamily* approximating the OSUB, for $h = \{10, 50, 100\}$, regarding the $\mathcal{A}$-metric and the computational time, for the Group 1.



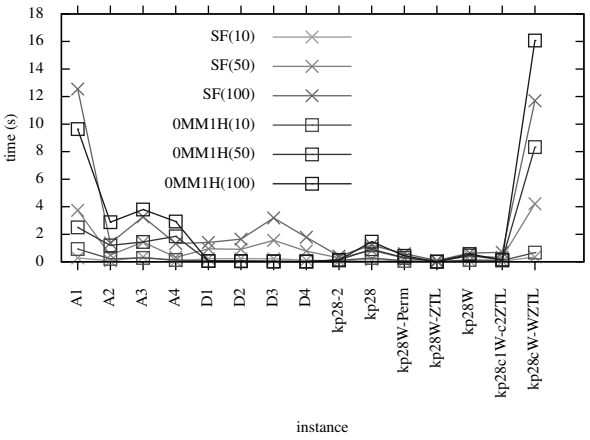
(a) Comparison of $\mathcal{A}$-metric for $h = 10$



(b) Comparison of $\mathcal{A}$-metric for $h = 50$

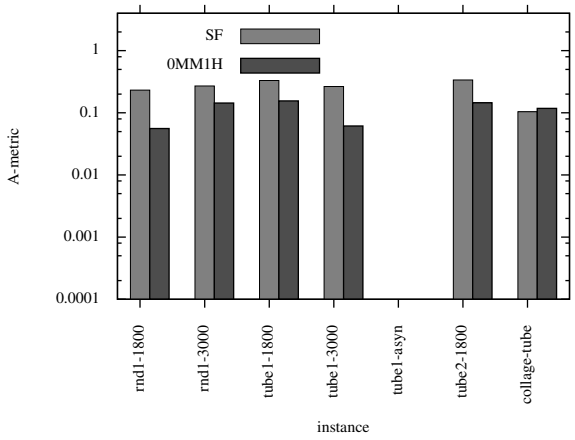


(c) Comparison of $\mathcal{A}$-metric for $h = 100$

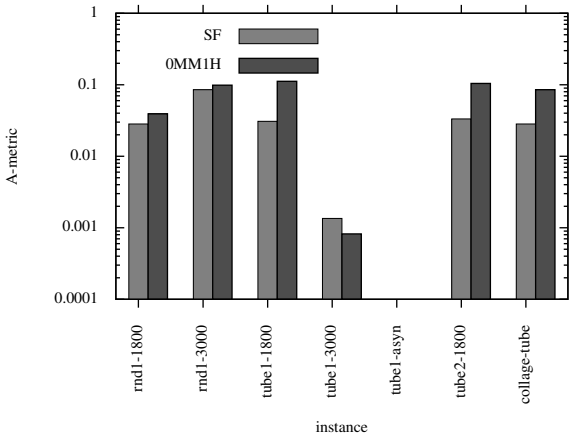


(d) Comparison of computational times

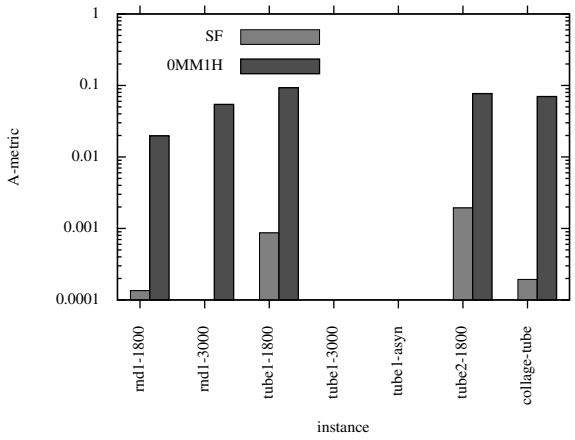
Figure B.3: Comparison of both heuristics *0MM1H + init* and *SurrogateFamily* approximating the OSUB, for $h = \{10, 50, 100\}$, regarding the $\mathcal{A}$-metric and the computational time, for the Group 2.



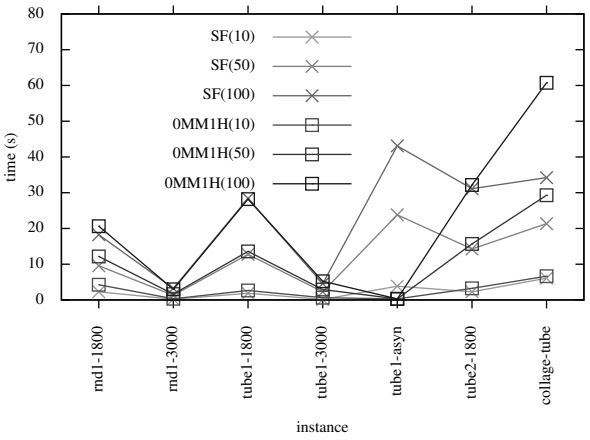
(a) Comparison of $\mathcal{A}$-metric for $h = 10$



(b) Comparison of $\mathcal{A}$-metric for $h = 50$



(c) Comparison of $\mathcal{A}$-metric for $h = 100$



(d) Comparison of computational times

Figure B.4: Comparison of both heuristics *0MM1H + init* and *SurrogateFamily* approximating the OSUB, for $h = \{10, 50, 100\}$, regarding the $\mathcal{A}$-metric and the computational time, for the Group 3.

List of Tables

List of Figures

Bibliography

Fotis Aisopos, Konstantinos Tserpes, and Theodora Varvarigou. Resource management in software as a service using the knapsack problem model. *International Journal of Production Economics*, 141(2):465 – 477, 2013. 47

Yash P. Aneja and K. P. K. Nair. Bicriteria transportation problem. *Management Science*, 25(1): 73–78, 1979. 34, 35, 43, 65, 96

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002. 85

Egon Balas and Eitan Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28:1130–1154, 1980. 50

Stefan Balev, Nicola Yanev, Arnaud Fréville, and Rumen Andonov. A dynamic programming based procedure for the multidimensional 0-1 knapsack problem. *European Journal of Operational Research*, 186(1):63–76, 2008. 53

Matthieu Basseur and Edmund K Burke. Indicator-based multi-objective local search. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3100–3107. IEEE, 2007. 45

Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Solving efficiently the 0-1 multi-objective knapsack problem. *Computers and Operations Research*, 36:260–279, 2009a. 43, 58, 62, 64, 155

Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Implementing an efficient FPTAS for the 0-1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1): 47–56, 2009b. 55

Tolga Bektas and Osman Oguz. On separating cover inequalities for the multidimensional knapsack problem. *Computers and Operations Research*, 34(6):1771–1776, 2007. 141

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957. 43

Harold P. Benson. An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem. *Journal of Global Optimization*, 13(1):1–24, 1998. 28

Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London, 1985. 83

Endre Boros. On the complexity of the surrogate dual of 0–1 programming. *Mathematical Methods of Operations Research*, 30(3):A145–A153, 1986. 53

Vincent Boyer. *Contribution à la programmation en nombres entiers*. PhD thesis, Université de Toulouse, 2007. 53

Vincent Boyer, Moussa Elkihel, and Didier El Baz. Heuristics for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 199(3):658 – 664, 2009. 52

Vincent Boyer, Didier El Baz, and Moussa Elkihel. Solving knapsack problems on GPU. *Computers and Operations Research*, 39(1):42–47, 2012. 51

M. Eugénia Captivo, João Clímaco, José R. Figueira, Ernesto Martins, and José Luis Santos. Solving bicriteria 0–1 knapsack problems using a labeling algorithm. *Computers and Operations Research*, 30(12):1865 – 1886, 2003. 43, 55, 66

Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4):305–337, 1973. 42

Vašek Chvátal. *Linear Programming*. Freeman, 1983. 22

Tommy Clausen, Allan Nordlunde Hjorth, Morten Nielsen, and David Pisinger. The off-line group seat reservation problem. *European Journal of Operational Research*, 207(3):1244 – 1253, 2010. 47

Halan Crowder, Ellis L. Johnson, and Manfred W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983. 13, 42, 140, 141, 152, 156

Carlos Gomes da Silva, João C. N. Clímaco, and José R. Figueira. A scatter search method for the bi-criteria multi-dimensional {0, 1}-knapsack problem using surrogate relaxation. *Journal of Mathematical Modelling Algorithms*, 3(3):183–208, 2004. 47, 58

Carlos Gomes da Silva, José R. Figueira, and João Clímaco. Integrating partial optimization with scatter search for solving bi-criteria {0, 1}-knapsack problems. *European Journal of Operational Research*, 177(3):1656–1677, 2007. 55

Carlos Gomes da Silva, João C. N. Clímaco, and José R. Figueira. Core problems in bi-criteria {0, 1}-knapsack problems. *Computers and Operations Research*, 35(7):2292–2306, 2008. 55

Kerstin Dächert and Kathrin Klamroth. A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization*, pages 1–34, 2014. 37

Luis DaCosta, Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 10th annual conference on genetic and evolutionary computation*, volume 5199, pages 913–920, 2008. 85

George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957. 49

Ismael R. de Farias Jr. and Georges L. Nemhauser. A polyhedral study of the cardinality constrained knapsack problem. *Mathematical Programming*, 96(3):439–467, 2003. 42

Kalyanmoy Deb. Multi-objective evolutionary algorithms. In *Springer Handbook of Computational Intelligence*, pages 995–1015. Springer, 2015. 45

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6 (2):182–197, 2002. 45

Fabien Degoutin and Xavier Gandibleux. Un retour d'expérience sur la résolution de problèmes combinatoires bi-objectifs. In *5e journée du groupe de travail Programmation Mathématique MultiObjectif (PM20)*, 2002. 66

Charles Delort. *Algorithmes d'énumération implicite pour l'optimisation multi-objectifs exacte : exploration d'ensembles bornant et application aux problèmes de sac à dos et d'affectation*. PhD thesis, Université Pierre et Marie Curie Paris VI, 2011. 11, 39, 42, 55, 56, 62, 91, 155

Charles Delort and Olivier Spanjaard. Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem. In Paola Festa, editor, *SEA*, volume 6049 of *Lecture Notes in Computer Science*, pages 253–265. Springer, 2010. 38, 41, 43, 55, 57, 65, 67, 126, 128

Jérémie Dubois-Lacoste. *Anytime Local Search for Multi-Objective Combinatorial Optimization: Design, Analysis and Automatic Configuration*. PhD thesis, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2014. 45

M.E. Dyer. Calculating surrogate constraints. *Mathematical Programming*, 19(1):255–278, 1980. ISSN 0025-5610. doi: 10.1007/BF01581647. 53

Matthias Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, 7(1):5–31, 2000. 28

Matthias Ehrgott. *Multicriteria Optimization*. Springer, Berlin, Heidelberg, 2 edition, 2005. 13, 28

Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum*, 22(4):425–460, 2000. 34, 45

Matthias Ehrgott and Xavier Gandibleux. Bounds and bound sets for biobjective combinatorial optimization problems. In *Multiple Criteria Decision Making in the New Millennium*, volume 507 of *Lecture Notes in Economics and Mathematical Systems*, pages 241–253. Springer Berlin Heidelberg, 2001. 29, 31, 171

Matthias Ehrgott and Xavier Gandibleux. Multiobjective combinatorial optimization. In *Multiple Criteria Optimization – State of the Art Annotated Bibliographic Surveys*, volume 52, pages 369–444. Kluwer Academic Publishers, Boston, MA, 2002. 34

Matthias Ehrgott and Xavier Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers and Operations Research*, 34(9):2674 – 2694, 2007. 29, 30, 31, 32, 96, 171

Matthias Ehrgott and Xavier Gandibleux. Hybrid metaheuristics for multi-objective combinatorial optimization. In *Hybrid metaheuristics*, pages 221–259. Springer, 2008. 45

Matthias Ehrgott and Dagmar Tenfelde-Podehl. Computation of ideal and nadir values and implications for their use in MCDM methods. *European Journal of Operational Research*, 151(1): 119–139, 2003. 29

Matthias Ehrgott, Andreas Löhne, and Lizhen Shao. A dual variant of benson's "outer approximation algorithm" for multiple objective linear programming. *Journal of Global Optimization*, 52 (4):757–778, 2012. 28

Thomas Erlebach, Hans Kellerer, and Ulrich Pferschy. Approximating multi-objective knapsack problems. In *Algorithms and Data Structures*, volume 2125 of *Lecture Notes in Computer Science*, pages 210–221. Springer Berlin Heidelberg, 2001. 44, 58

Gerald W. Evans. An overview of techniques for solving multiobjective mathematical programs. *Management Science*, 30(11):1268–1282, 1984. 27

José R. Figueira, Gabriel Tavares, and Margaret M. Wiecek. Labeling algorithms for multiple objective integer knapsack problems. *Computers and Operations Research*, 37(4):700–711, 2010. 55

José R. Figueira, Luís Paquete, Marco Simões, and Daniel Vanderpooten. Algorithmic improvements on dynamic programming for the bi-objective 0,1 knapsack problem. *Computational Optimization and Applications*, 56(1):97–111, 2013. 43, 55

José R. Figueira, Pedro Correia, and Luís Paquete. Techniques for solution set compression in multiobjective optimization. In *Proceedings of the 23rd International Conference on Multiple Criteria Decision Making*, 2015. 55

Kostas Florios, George Mavrotas, and Danae Diakoulaki. Solving multiobjective, multiconstraint knapsack problems using mathematical programming and evolutionary algorithms. *European Journal of Operational Research*, 203(1):14–21, 2010. 10, 39, 41, 58, 63, 155

Arnaud Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1 – 21, 2004. 48, 52

Arnaud Fréville and Gérard Plateau. An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem. *European Journal of Operational Research*, 68(3): 413–421, 1993. 12, 52, 53, 54, 59, 93, 95, 96, 155

Arnaud Fréville and Gérard Plateau. An efficient preprocessing procedure for the multidimensional 0- 1 knapsack problem. *Discrete Applied Mathematics*, 49(1-3):189–212, 1994. 53

Arnaud Fréville and Gérard Plateau. The 0-1 bidimensional knapsack problem: Toward an efficient high-level primitive tool. *Journal of Heuristics*, 2(2):147–167, 1996. 52

Virginie Gabrel and Michel Minoux. A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems. *Operations Research Letters*, 30(4): 252–264, 2002. 142

Sune L. Gadegaard, Matthias Ehrgott, and Lars R. Nielsen. A cut-and-branch approach for a class of bi-objective combinatorial optimization problems. In *23rd International Conference on Multiple Criteria Decision Making*, 2015. 43

Xavier Gandibleux and Arnaud Fréville. Tabu search based procedure for solving the 0-1 multi-objective knapsack problem: The two objectives case. *Journal of Heuristics*, 6:361–383, 2000. 46, 55, 56, 57, 66

Xavier Gandibleux and Olga Perederieieva. Some observations on the bi-objective 01 bi-dimensional knapsack problem, 2011. 10, 58, 59, 63, 93, 94, 155, 156

Xavier Gandibleux, Hiroyuki Morita, and Naoki Katoh. The supported solutions used as a genetic information in a population heuristic. In *Evolutionary Multi-Criterion Optimization*, pages 429–442. Springer, 2001. 131

Michael R. Garey and David S. Johnson. Computers and intractability: a guide to NP-completeness, 1979. 23

Bezalel Gavish and Hasan Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31:78–105, 1985. 52, 53

George Gens and Eugene Levner. Computational complexity of approximation algorithms for combinatorial problems. In *Mathematical Foundations of Computer Science 1979*, volume 74 of *Lecture Notes in Computer Science*, pages 292–300. Springer, 1979. 52

Arthur M. Geoffrion. Proper Efficiency and the Theory of Vector Maximization. *Journal of Mathematics, Analysis and Applications*, 22(3):618–630, 1968. 26, 34

Fred Glover. A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, 13(6):879–919, 1965. 33, 56

Fred Glover. Surrogate Constraint Duality in Mathematical Programming. *Operations Research*, 23:434–451, 1975. 33

Fred Glover. Advanced greedy algorithms and surrogate constraint methods for linear and quadratic knapsack and covering problems. *European Journal of Operational Research*, 230 (2):212 – 225, 2013. 52

Fred Glover and Gary A Kochenberger. *Handbook of metaheuristics*. Springer Science and Business Media, 2003. 45, 131

Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 39:653–684, 2000. 131

David E Goldberg. Probability matching, the magnitude of reinforcement, and classifier system bidding. *Machine Learning*, 5(4):407–425, 1990. 81

R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278, 1958. 42

Yacov Y. Haimes, Leon S. Lasdon, and David A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1:296–297, 1971. 35

Saïd Hanafi and Arnaud Fréville. An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 106:659–675, 1998. 52, 136

Pierre Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Berlin Heidelberg, 1980. 25

Mhand Hifi and Hedi Mhalla. Sensitivity analysis to perturbations of the weight of a subset of items: The knapsack case study. *Discrete Optimization*, 10(4):320–330, 2013. 51

Raymond R. Hill, Yong Kun Cho, and James T. Moore. Problem reduction heuristic for the 0-1 multidimensional knapsack problem. *Computers and Operations Research*, 39(1):19–26, 2012. 53

Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004. 45

IBM. Website of CPLEX, 2015. URL http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/. 42

Giorgio P Ingargiola and James F Korsh. Reduction algorithm for zero-one single knapsack problems. *Management science*, 20(4-part-i):460–463, 1973. 51, 53, 126

Hisao Ishibuchi, Yasuhiro Hitotsuyanagi, Noritaka Tsukamoto, and Yusuke Nojima. Implementation of multiobjective memetic algorithms for combinatorial optimization problems: A knapsack problem case study. In *Multi-Objective Memetic Algorithms*, volume 171 of *Studies in Computational Intelligence*, pages 27–49. Springer Berlin Heidelberg, 2009. 58

Mohammad Jalali Varnamkhasti. Overview of the algorithms for solving the multidimensional knasack problem. *Advanced Sutdies in Biology*, 4(1):37–47, 2012. 48, 52

Andrzej Jaszkiewicz. On the computational efficiency of multiple objective metaheuristics. the knapsack problem case study. *European Journal of Operational Research*, 158(2):418 – 433, 2004. 58

Julien Jorge. *Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires*. PhD thesis, Université de Nantes, 2010. 10, 11, 39, 41, 42, 43, 55, 56, 57, 62, 65, 67, 91, 114, 126, 127, 155

Nicolas Jozefowiez, Gilbert Laporte, and Frédéric Semet. A generic branch-and-cut algorithm for multiobjective optimization problems: Application to the multilabel traveling salesman problem. *INFORMS Journal on Computing*, 24(4):554–564, 2012. 10, 42, 155

Mark H. Karwan and Ronald L. Rardin. Some relationships between lagrangian and surrogate duality in integer programming. *Mathematical Programming*, 17(1):320–334, 1979. ISSN 0025-5610. doi: 10.1007/BF01588253. 53

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. 47, 48, 51

Mahdi Khemakhem, Boukthir Haddar, Khalil Chebil, and Saïd Hanafi. A filter-and-fan metaheuristic for the 0-1 multidimensional knapsack problem. *International Journal of Applied Metaheuristic Computing*, 3(4):43–63, 2012. 52

Gokhan Kirlik and Serpil Sayın. Computing the nadir point for multiobjective discrete optimization problems. *Journal of Global Optimization*, 2014. 29

Gokhan Kirlik and Serpil Sayın. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232 (3):479 – 488, 2014. 37

Gülseren Kiziltan and Erkut Yucaoğlu. An algorithm for multiobjective zero-one linear programming. *Management Science*, 29(12):1444–1453, 1983. 38, 39, 42

Diego Klabjan, Georges L. Nemhauser, and Craig A. Tovey. The complexity of cover inequality separation. *Operations Research Letters*, 23(1-2):35–40, 1998. 141

Kathrin Klamroth and Margaret M. Wiecek. Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics*, pages 57–76, 2000. 43, 55

Marco Laumanns, Lothar Thiele, and Eckart Zitzler. An adaptive scheme to generate the pareto front based on the epsilon-constraint method. *Practical Approaches to Multi-Objective Optimization*, 4461, 2005. 37

Banu Lokman and Murat Köksalan. Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, 57(2):347–365, 2013. 37

Helena R Lourenço, Olivier C Martin, and Thomas Stützle. *Iterated local search*. Springer, 2003. 45

Abilio Lucena and John E. Beasley. Branch and cut algorithms. *Advances in Linear and Integer Programming Oxford University Press*, pages 187—221, 1996. 42

Thibaut Lust and Jacques Teghem. Two-phase pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16(3):475–510, June 2010. 45

Thibaut Lust and Jacques Teghem. The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research*, 2012. 48, 58

Andrew O. Makhorin. Website of GLPK, 2015. URL `http://www.gnu.org/software/glpk/`. 42

Vittorio Maniezzo, Thomas Stützle, and Stefan Voß, editors. *Matheuristics - Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, 2010. 46

Silvano Martello and Paolo Toth. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, 1(3):169 – 175, 1977. 49, 50, 55, 56

Silvano Martello and Paolo Toth. *Knapsack Problems : Algorithms and Computer Implementations*. John Wiley & sons, 1990. 47

Silvano Martello and Paolo Toth. An exact algorithm for the two-constraint 0-1 knapsack problem. *Operations Research*, 51(5):pp. 826–835, 2003. 52

Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999. 51, 53, 55, 65, 114

Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984. 43

Jorge Maturana, Alvaro Fialho, Frédéric Saubion, Marc Schoenauer, Frédéric Lardeux, and Michele Sebag. Adaptive operator selection and management in evolutionary algorithms. In *Autonomous Search*, pages 161–189. Springer, 2012. 79

George Mavrotas, José R. Figueira, and Kostas Florios. Solving the bi-objective multi-dimensional knapsack problem exploiting the concept of core. *Applied Mathematics and Computation*, 215 (7):2502 – 2514, 2009. 58

George Mavrotas, José R. Figueira, and Alexandros Antoniadis. Using the idea of expanded core for the exact solution of bi-objective multi-dimensional knapsack problems. *Journal of Global Optimization*, 49(4):589–606, 2011. 58

María Auxilio Osorio and Germán Cuaya. Hard problem generation for MKP. In *Proceedings of the 6th Mexican International Conference on Computer Science*, ENC '05, pages 290–297, Washington, DC, USA, 2005. IEEE Computer Society. 115, 169

María Auxilio Osorio, Fred Glover, and Peter Hammer. Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. *Annals of Operations Research*, 117(1-4): 71–93, 2002. 53

Melih Özlen, Benjamin A. Burton, and Cameron A. G. MacRae. Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications*, 160 (2):470–482, 2014. 37

Özgür Özpeynirci and Murat Köksalan. An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, 56(12): 2302–2315, 2010. 35

Manfred W. Padberg and Giovanni Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1 – 7, 1987. 42

ES Page. Continuous inspection schemes. *Biometrika*, pages 100–115, 1954. 85

Luis Paquete and Thomas Stützle. A two-phase local search for the biobjective traveling salesman problem. In *Evolutionary Multi-Criterion Optimization*, pages 479–493. Springer, 2003. 45

Vilfredo Pareto. *Manuel d'économie politique*. F. Rouge, Lausanne, 1896. 24

Olga Perederieieva. Bound sets for bi-objective bi-dimensional knapsack problem. Master thesis, Université de Nantes, 2011. 94, 95, 112, 114

David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45: 758–767, 1994. 51

Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, 185(2):509–533, 2008. 43

Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete optmization*, 7(3):149–165, 2010a. 43

Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *INFORMS Journal on Computing*, 22(3):371–386, 2010b. 13, 35, 156

Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy. The core concept for the multidimensional knapsack problem. In *Evolutionary Computation in Combinatorial Optimization*, volume 3906 of *Lecture Notes in Computer Science*, pages 195–208. Springer Berlin Heidelberg, 2006. 52

Rosa M. Ramos, S. Alonso, J. Sicilia, and C. Gonzalez. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111(3):617–628, 1998. 38, 41

Colin R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Inc., New York, NY, USA, 1993. 45

Hershel M. Safer and James B. Orlin. Fast approximation schemes for multi-criteria combinatorial optimization. Technical Report 3756-95, Sloan School of Management, 1995a. 45

Hershel M. Safer and James B. Orlin. Fast approximation schemes for multi-criteria flow, knapsack, and scheduling problems. Technical Report 3757-95, Sloan School of Management, 1995b. 45, 55

Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, Chichester, 1986. 22

Britta Schulze and Kathrin Klamroth. Multiple objective optimization for multidimensional knapsack problems. In *23rd International Conference on Multiple Criteria Decision Making*, 2015. 52

Britta Schulze, Kathrin Klamroth, and Luis Paquete. Transforming constraints into objectives: experiments with bidimensional knapsack problems. In *22nd International Conference on Multiple Criteria Decision Making*, 2013. 52

Wei Shih. A branch and bound method for the multiconstraint zero-one knapsack problem. *The Journal of the Operational Research Society*, 30(4):pp. 369–378, 1979. 52

Francis Sourd and Olivier Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20:472–484, 2008. 38, 39, 40, 41, 42

Thomas Stidsen and Kim Allan Andersen. Optimized parallel branch and cut algorithm for biobjective tsp. Workshop Recent Advances in Multi-Objective Optimization, 2015. 43

Jacques Teghem, editor. *Programmation linéaire (2ème édition)*. Elsevier, ellipses et editions de l'ULB edition, 2003. 379 pages. 22

Arne Thesen. A recursive branch and bound algorithm for the multidimensional knapsack problem. *Naval Research Logistics Quarterly*, 22(2):341–353, 1975. 52

Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *IEEE Congress on Evolutionary Computation*, pages 1539–1546. IEEE, 2005. 82

Fabien Tricoire. Multi-directional local search. *Computers and Operations Research*, 39(12): 3089–3101, 2012. 58

George Tsaggouris and Christos Zaroliagis. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. In Tetsuo Asano, editor, *Algorithms and Computation*, volume 4288 of *Lecture Notes in Computer Science*, pages 389–398. Springer Berlin Heidelberg, 2006. 45

Ekunda Lukata Ulungu and Jacques Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995. 43

Ekunda Lukata Ulungu and Jacques Teghem. Solving multi-objective knapsack problem by a branch-and-bound procedure. *Multicriteria Analysis*, pages 269–278, 1997. 39, 42, 55, 62

Igor L. Vasil'ev. A cutting plane method for knapsack polytope. *Journal of Computer and Systems Sciences International*, 48(1):70–77, 2009. 42

Bernardo Villarreal and Mark H. Karwan. An interactive dynamic programming approach to multicriteria discrete programming. *Journal of Mathematical Analysis and Applications*, 81(2): 524 – 544, 1981. 29, 31, 171

Thomas Vincent. *Caractérisation des solutions efficaces et algorithmes d'énumération exacts pour l'optimisation multiobjectif en variables mixtes binaires*. PhD thesis, Université de Nantes, 2013. 43

Thomas Vincent, Florian Seipp, Stefan Ruzika, Anthony Przybylski, and Xavier Gandibleux. Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biojective case. *Computers and Operations Research*, 40(1):498–509, 2013. 38

Marc Visée, Jacques Teghem, Marc Pirlot, and Ekunda Lukata Ulungu. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, 12:139–155, 1998. 39, 42, 43, 55, 57, 62, 66, 126

H. Martin Weingartner and David N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1):pp. 83–103, 1967. 52

Laurence A. Wolsey. *Integer programming*. Wiley-Interscience, 1998. 23, 32, 33, 42

Laurence A. Wolsey and Georges L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1999. 23

Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4): 257–271, 1999. 58

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Audrey CERQUEUS

Algorithmes de *branch-and-cut* bi-objectif appliqués au problème de sac-à-dos en variables binaires

ensembles bornants surrogate, stratégies de branchement dynamiques, génération et exploitation d'inégalités de couverture.

Bi-objective branch-and-cut algorithms applied to the binary knapsack problem

surrogate bound sets, dynamic branching strategies, generation and exploitation of cover inequalities.

Résumé

Dans ce travail, nous nous intéressons à la résolution de problèmes d'optimisation combinatoire multi-objectif. Ces problèmes ont suscité un intérêt important au cours des dernières décennies. Afin de résoudre ces problèmes, particulièrement difficiles, de manière exacte et efficace, les algorithmes sont le plus souvent spécifiques au problème traité. Dans cette thèse, nous revenons sur l'approche dite de *branch-and-bound* et nous en proposons une extension pour obtenir un *branch-and-cut*, dans un contexte bi-objectif. Les problèmes de sac-à-dos sont utilisés comme support pour ces travaux. Trois axes principaux sont considérés : la définition de nouveaux ensembles bornants, l'élaboration d'une stratégie de branchement dynamique et la génération d'inégalités valides. Les ensembles bornants définis sont basés sur la relaxation surrogate, utilisant un ensemble de multiplicateurs. Des algorithmes sont élaborés, à partir de l'étude des différents multiplicateurs, afin de calculer efficacement les ensembles bornants surrogate. La stratégie de branchement dynamique émerge de la comparaison de différentes stratégies de branchement statiques, issues de la littérature. Elle fait appel à une méthode d'apprentissage par renforcement. Enfin, des inégalités de couverture sont générées et introduites, tout au long de la résolution, dans le but de l'accélérer. Ces différents apports sont validés expérimentalement et l'algorithme de *branch-and-cut* obtenu présente des résultats encourageants.

Mots clés

optimisation combinatoire multi-objectif, branch-and-cut, stratégie de branchement, relaxation surrogate, problème de sac-à-dos bi-objectif.

Abstract

In this work, we are interested in solving multi-objective combinatorial optimization problems. These problems have received a large interest in the past decades. In order to solve exactly and efficiently these problems, which are particularly difficult, the designed algorithms are often specific to a given problem. In this thesis, we focus on the branch-and-bound method and propose an extension by a branch-and-cut method, in bi-objective context. Knapsack problems are the case study of this work. Three main axis are considered: the definition of new upper bound sets, the elaboration of a dynamic branching strategy and the generation of valid inequalities. The defined upper bound sets are based on the surrogate relaxation, using several multipliers. Based on the analysis of the different multipliers, algorithms are designed to compute efficiently these surrogate upper bound sets. The dynamic branching strategy arises from the comparison of different static branching strategies from the literature. It uses reinforcement learning methods. Finally, cover inequalities are generated and introduced, all along the solving process, in order to improve it. Those different contributions are experimentally validated and the obtained branch-and-cut algorithm presents encouraging results.

Key Words

multi-objective combinatorial optimization, branch-and-cut, branching strategies, surrogate relaxation, bi-objectif knapsack problem.