PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université Paris-Dauphine

The Budgeted Spanning Tree Problem: Extended formulations, Projection, and Branch-and-Cut

Soutenue par
Charles NOURRY
Le 11 juin 2025

École doctorale n°543
ED SDOSE

Spécialité
Informatique

Composition du jury :

Mourad BAIOU French National Centre for Scientific Research	*Président*
François CLAUTIAUX Institut de Mathématiques de Bordeaux	*Rapporteur*
Frédéric MEUNIER École nationale des ponts et chaussées	*Rapporteur*
Matthieu CHARDY Orange Labs	*Examinateur*
Adam N. LETCHFORD Lancaster University	*Examinateur*
Hande YAMAN Faculty of Economics and Business	*Examinatrice*
Ali Ridha MAHJOUB Université Paris-Dauphine	*Directeur de thèse*
Hassène AISSI Université Paris-Dauphine	*Co-encadrant*

Dauphine | PSL UNIVERSITÉ PARIS

Remerciements

Cette thèse constitue l'aboutissement d'un travail collectif. Au travers de ces quelques lignes, je souhaite remercier les personnes qui y ont contribuées, directement ou indirectement.

Je tiens à remercier Monsieur A. Ridha Mahjoub, Professeur émérite à l'Université Paris-Dauphine, pour avoir accepté d'être mon directeur de thèse et pour m'avoir accompagné pendant toutes ces années. Ridha m'a transmis sa passion pour les approches polyédrales et la programmation linéaire au cours de mes années de Master où j'ai eu la chance d'être son étudiant. Je le remercie beaucoup pour la confiance qu'il m'a accordée en acceptant de poursuivre avec moi en thèse. Je lui suis profondément reconnaissant d'avoir partagé avec moi son expérience et ses connaissances, et le remercie encore pour sa disponibilié, son sérieux, son énergie et ses paroles motivantes et réconfortantes pendant les périodes plus difficiles.

Je remercie également Monsieur Hassène Aissi, Maître de conférences à l'Université Paris-Dauphine, d'avoir accepté de co-encadré ma thèse. Je tiens à remmercier Hassène pour son investissement conséquent pendant toutes ces années, notamment dans la rédaction de ce manuscrit. Je lui suis profondément reconnaissant de m'avoir inculquer les codes du monde de la recherche scientifique ainsi que d'avoir partagé avec moi son expérience et ses connaissances dans de nombreux domaines. Je le remercie encore pour le temps et l'énergie qu'il m'a consacré pendant ces années, ce travail n'aurait pas été le même sans lui.

Je remercie Monsieur François Clautiaux, Professeur à l'Institut de Mathématiques de Bordeaux, d'avoir accepté de rapporter ma thèse. Je le remercie pour l'intérêt qu'il a porté à mon travail ainsi que ses suggestions pertinentes.

Je remercie également Monsieur Frédéric Meunier, Professeur à l'École Nationale des Ponts et Chaussées, pour m'avoir fait l'honneur de rapporter ma thèse. Je le remercie également pour sa lecture approfondie du manuscrit et ses remarques constructives sur mes travaux.

Je remercie Monsieur Mourad Baïou, Directeur de recherche au CNRS, d'avoir accepté de présider mon jury. Je le remercie pour sa lecture précise du manuscrit et pour ses commentaires pertinents.

Je remercie Monsieur Matthieu Chardy, Ingénieur de recherche à Orange Labs, d'avoir accepté de participer à mon jury. Je voudrais en profiter également pour remercier

chaleureusement mon ancien maître de stage pour m'avoir transmis son expérience et ses connaissances lors de ce dernier. Cet aprentissage s'est révélé particulièrement précieux pendant ma thèse.

Je remercie également Monsieur Adam N. Letchford, Professeur à Lancaster University, de m'avoir fait l'honneur de participer à mon jury. Je le remercie également pour sa lecture minutieuse du manuscrit et ses commentaires.

Je remercie Madame Hande Yaman, Professeure à Faculty of Economics and Business, d'avoir accepté de participer à mon jury. Je la remercie pour l'intérêt qu'elle a portée à mon travail ainsi que pour ses commentaires intéressants lors de la soutenance.

Je tiens à exprimer chaleureusement ma reconnaissance à mes collègues et amis lamsadiens que j'ai eu la chance de rencontrer, mes mots ne suffisent pas à leurs exprimer la reconnaissance que j'ai pour eux, pour le soutien qu'ils m'ont apporté, directement ou indirectement, Je tiens à remercier mes frères et soeurs de galères; Mehdi, Hiba, Virginia, Jerome, Manel, Lucas, Céline, Eric, Isma, Noémie, Houria, George, Emma, Pierre, Bastien, Pierre, Thibault, Théo, Virginie, Boris, Louise, Amin, Axel, Nicolas, Felix, Felipe, Skander, Lucas, Matthieu, Marie, Diana, Bochra, Sébastien, Julien, Hossein, Mehdi, Louis, Khalil, Ahlam, Iskander, Lola, Yassine, Nikos, Yassine, Beatrice, Ons, Rafael, Gil, Ahmad, Oussama, Ariane, Camille, Sarra, Scarlett, Raja, Berkay, Manolis, Sofia, Ikko, Jiaxin, Jinfeng mais également les personnes que j'aurais oublié de mentionner. Je remercie plus particulièrement les personnes avec lesquelles j'ai eu l'honneur de partager le bureau C605 ainsi que les collaborateurs de playlists et de cadeaux qui m'ont beaucoup apportées et que je chéris particulièrement.

Je remercie également tous les membres permanents du LAMSADE et de l'Université Paris Dauphine. Je remercie plus chaleureusement Marie-Clotilde, Sonia, Olivier, Satya, Hugo, Jérome, Joyce et Christian pour les bons moments passés ensemble et pour leurs soutiens. Je remercie également Rida, Laurent, les membres de mon CSI, de l'école doctorale ainsi que du CROUS pour leurs sérieux et leurs efficacités.

Je tiens à remercier chaleureusement les membres du groupe de travail Polyèdres et Optimisation Combinatoire pour l'organisation de journées auxquelles je garderais de magnifiques souvenirs ainsi que pour la convivialité de nos échanges. Je tiens à remercier plus particulièrement Pierre Fouilhoux, Jean Mailfert, Fatiha Bendali, Denis Cornaz, Emiliano Lancini, Eduardo Uchoa et Ivana Ljubic.

Je tiens à remercier Ines qui m'a motivée à contacter Ridha Mahjoub afin de lui parler de mes envies de travailler avec lui. De nombreuses choses auraient été différentes sans elle.

Pour finir, mes sincères remerciements vont également à ma famille et à mes amis, pour leurs présences et leurs soutiens. Merci de m'avoir permis de me changer les idées quand j'en avais besoin. Merci à tout ceux qui ont arrêté de me poser la question fatidique "C'est pour quand cette thèse ?", mais également à tout ceux qui ont continué de la poser. Je vous aime.

Abstract

The d-Budgeted Spanning Tree Problem lies at the intersection of the spanning tree and the multi-knapsack problems. It models several applications in the fields of network design, power and transportation networks. The multi-knapsack part allows to consider several criteria, potentially in conflict, that occur in many real-world problems. We consider exact algorithms, based on a polyhedral approach, to solve the problem with one or several budget constraints. In the case where d = 1, we give an exact extended formulation based on matroids intersection and disjunctive programming. Due to the NP-hardness of the problem, this formulation is exponentially large and prohibitive to solve. Moreover, there is no hope to obtain all the facets of the convex hull by projection. Instead, we carefully select rays of the projection cone that exploit the spanning tree and the knapsack polytopes. We develop a Branch-and-Cut algorithm based on a careful selection of rays of the projection cone, efficient separation routines, and a fast algorithm to solve the Lagrangian relaxation of the problem. Our experiment results show that our approach outperforms state of the art ones. We generalize our algorithm to solve the problem with $d \geq 2$ budget constraints.

Keywords: Combinatorial optimization, Polyhedra, Spanning tree, Knapsack, Matroid, Extended formulation, Projection, Valid inequality, Lagrangian relaxation, Branch-and-Cut.

Résumé long

Le problème de l'arbre couvrant d-budgété : formulations étendues, projection et algorithmes de coupes et branchements

L'optimisation combinatoire est une branche de l'informatique et des mathématiques appliquées. Un problème d'optimisation combinatoire consiste à trouver un des meilleurs éléments parmi un ensemble fini de solutions. La plupart de ces problèmes ne peuvent pas être résolus en temps polynomial (sauf si $P = NP$) et la conception d'algorithmes efficaces constitue de véritables challenges. L'étude et l'exploitation de la structure des problèmes d'optimisation combinatoire difficiles sont primordiaux dans l'élaboration de méthodes exactes. Les approches polyédrales, initiées en 1965 par Edmonds [23], s'avèrent particulièrement efficaces dans la résolution de problèmes d'optimisation combinatoire difficiles. Celles-ci consistent à ramener un problème sous la forme d'un système d'inégalités linéaires décrivant l'enveloppe convexe de ses solutions. Une analyse fine de ce polyèdre permet généralement d'obtenir des procédures performantes pour résoudre le problème.

Des problèmes d'optimisation combinatoire sont quotidiennement traités par des entreprises, collectivités territoriales, administrations et individus. Le problème de l'arbre couvrant de coût minimal est un problème d'optimisation combinatoire classique. Ce dernier consiste à déterminer un sous-réseau acyclique connectant tous les sommets d'un réseau pondéré. Ce problème est au coeur de nombreuses applications, notamment dans les domaines des télécommunications, de l'énergie, des transports. Le problème de l'arbre couvrant de coût minimal peut être résolu en temps polynomial (algorithmes de Prim, Kruskal, Borůvka). Le problème du sac-à-dos est également un célèbre problème d'optimisation combinatoire. On dispose d'un sac-à-dos d'une capacité finie ainsi qu'un ensemble d'éléments ayant chacun un volume et un profit. L'objectif étant de déterminer un sous-ensemble d'élements à placer dans le sac tel que le profit total soit maximum. Ce problème modélise l'allocation de ressources, la gestion de stocks comme la préparation d'une valise pour un voyage. Contrairement à l'arbre couvrant, il n'existe pas d'algorithme polynomial pour résoudre le problème de sac-à-dos. Le problème du sac-à-dos multidimensionnel est une variante du sac-à-dos dans lequel les éléments et le sac-à-dos ont des volumes dans plusieurs dimensions.

Dans cette thèse, nous considérons le problème de l'arbre couvrant d-budgété, celui-ci combine les problèmes de l'arbre couvrant de coût minimal, une partie bien structurée, et du sac-à-dos multidimensionnel, une partie dépendente de la pondération des coûts et

des poids de dimension d. La difficulté du problème réside dans le caractère arbitraire des valeurs des coûts et des poids de la partie sac-à-dos multidimensionnel. Nous proposons plusieurs contributions théoriques pour le problème, telles que des formulations étendues exactes dans certains cas particuliers, des descripions du polyèdre obtenues via projection et une analyse faciale. Nous présentons également de nombreuses contributions pratiques basées sur notre approche polyédrale, constituant en des algorithmes exactes pour la résolution du problème de l'arbre couvrant d-budgété, dans les cas où une ou plusieurs contraintes de budget sont impliquées, mais aussi des algorithmes de séparation, des routines efficaces et des algorithmes de résolution de la relaxation Lagrangienne.

Par la suite, nous présentons succintement le contenu et les résultats majeurs de chaque chapitre.

1 Introduction

Dans le premier chapitre, nous présentons le problème de l'arbre couvrant d-budgété ainsi qu'un état de l'art couvrant des travaux proches au problème. Parmi ces derniers, nous considérons le problème de l'arbre couvrant multi-objectifs. Une relation forte peut être établie entre ces deux problèmes étant donné que l'on peut obtenir une version d-budgétée à partir d'un problème avec $d+1$ objectifs en transformant d fonctions objectifs en contraintes de sac-à-dos. En optimisation multi-objectifs, la résolution de problèmes d-budgétés est au coeur de nombreux algorithmes pour générer des solutions non-dominées au problème avec $d+1$ objectifs. La suite de l'état de l'art porte sur le problème de l'arbre couvrant d-budgété, dans les cas où $d = 1$ et $d \geq 2$. Le cas $d = 1$ a été sujet à de nombreux travaux, il en résulte plusieurs algorithmes d'approximation et quelques algorithmes exactes. Contrairement au cas précédent, la généralisation du problème de l'arbre couvrant d-budgété avec $d \geq 2$ n'a été que peu étudiée. Nous présentons donc des travaux portant sur des variantes telle que le problème de l'arbre couvrant avec contraintes de degré. Puis, nous positionnons nos contributions par rapport à l'état de l'art existant. Enfin, a la fin du chapitre, nous donnons une liste de notations qui seront utilisées tout au long du document.

2 Notions de base

Le second chapitre est consacré à la présentation de plusieurs définitions et notions de bases qui seront utiles pour l'ensemble du document. Dans un premier temps, nous introduisons des notions fondamentales telles que l'optimisation combinatoire, la complexité des algorithmes, les approches polyédrales, les algorithmes de branchements et la théorie des graphes. Enfin, dans un second temps nous introduisons des définitions et des techniques qui seront exploitées dans les chapitres clés de la thèse, telles que les formulations étendues, la programmation disjonctive et les matroïdes.

3 Le problème de l'arbre couvrant budgété minimal et polyèdre

Dans le troisième chapitre, nous nous intéressons à la description du polyèdre du problème de l'arbre couvrant d-budgété ainsi qu'à plusieurs formulations en nombres entiers du problème. Nous comparons plusieurs formulations venant de la littérature afin d'identifier les plus efficaces en théorie et en pratique. Les formulations basées sur l'élimination des sous-tours (3.5)-(3.8) ou la description de Miller-Tucker-Zemlin (3.15)-(3.20) semblent les plus pertinentes. Par la suite, nous analysons le polyèdre du problème de l'arbre couvrant d-budgété. En exploitant la structure du graphe ainsi que les poids des arêtes, nous donnons plusieurs bornes sur la dimension du polyèdre ainsi qu'une condition suffisante où la dimension est connue. Nous présentons également des conditions selon lesquelles les familles d'inégalités de la formulation d'élimination des sous-tours définissent des facettes pour le polyèdre de l'arbre couvrant d-budgété. Enfin, nous considérons le cas particulier où $d = 1$ et où la contrainte de budget définie un matroïde. Sous ces hypothèses, le polyèdre de l'arbre couvrant d-budgété est décrit par la formulation d'élimination des sous-tours.

4 Relaxations linéaire et Lagrangienne pour le problème de l'arbre couvrant 1-budgété

Le quatrième chapitre porte sur l'étude et la résolution des relaxations linéaire et Lagrangienne pour le problème de l'arbre couvrant 1-budgété. Dans ce cas particulier, une solution optimale de la relaxation linéaire de la formulation d'élimination des sous-tours constitue une solution 2-approchée pour le problème 1-budgété. Ce résultat est à la source de nombreux algorithmes d'approximation dans la littérature, générant des solutions de coûts très proches à celle d'une solution optimale au problème exact. Cependant, en pratique la séparation des inégalités de sous-tours peut prendre du temps. Nous nous intéressons donc à la relaxation Lagrangienne, une autre technique de relaxation mathématique consitant à supprimer une ou plusieurs contraintes du problème en les intégrant à la fonction objectif. Dans le cas de notre problème de l'arbre couvrant 1-budgété, en dualisant la contrainte de sac-à-dos nous obtenons un problème dont la valeur d'une solution optimale est équivalente celle de la relaxation linéaire. En exploitant la géométrie de la relaxation Lagrangienne, nous proposons sous la forme de l'Algorithme 4, un algorithme efficace calculant une solution optimale au problème de la relaxation Lagrangienne et de valeur proche d'une solution optimale au problème exact. Notre algorithme combine une recherche dichotomique, basée sur la recherche paramétrique de Megiddo [51] ainsi que des stratégies de pivots afin de converger rapidement vers une solution optimale. Nous présentons également un pré-traitement efficace basé sur la géométrie du problème paramétré afin d'accélérer la procédure. Les résultats expérimentaux présentés dans le Tableau 1 montrent que la résolution de la relaxation linéaire ne peut rivaliser avec notre algorithme. Dans une dernière partie, nous adaptons notre algorithme dans le cas particulier où les coûts sont binaires. Sous ces hypothèses, notre algorithme permet de résoudre le problème de l'arbre couvrant 1-budgété en temps polynomial.

Nombre de sommets des instances	Temps de résolution (s) Relaxation linéaire formulation des sous-tours	Temps de résolution (s) Algorithme 4	Temps de résolution (s) Algorithme 4 avec pré-traitement
100	0.348	0.021	0.017
200	6.767	0.112	0.088
300	60.082	1.023	0.347
400	228.420	2.731	0.819
500	657.672	4.235	1.217
600	3600.000	6.014	1.893
700	3600.000	9.002	2.782
800	3600.000	11.315	3.967
900	3600.000	14.539	4.852
1000	3600.000	18.223	5.671
1500	3600.000	40.003	13.044
2000	3600.000	73.440	23.180

Table 1: Temps de comparaison entre la résolution de la relaxation linéaire et Lagrangienne

5 Formulations étendues pour le problème de l'arbre couvrant budgété

Dans le cinquième chapitre, nous présentons des formulations étendues entières pour le problème de l'arbre couvrant 1-budgété ainsi que pour la version d-budgété dans le cas où le graphe est un cactus. Notre formulation étendue entière pour le problème de l'arbre couvrant 1-budgété est basée sur le théorème d'intersection de matroïdes d'Edmonds [24] ainsi que la programmation disjonctive de Balas [8]. Nous partitionnons le problème 1-budgété en plusieurs sous-problèmes d'intersection de matroïdes, dont chacun peut être associé à un polyèdre entier selon Edmonds. Puis, en nous appuyant sur les travaux de Balas, nous considérons un unique polyèdre décrivant la disjonction de tous les polyèdres entiers des sous-problèmes. Nous obtenons un polyèdre entier dont la projection sur l'espace des variables initiales correspond au polyèdre du problème de l'arbre couvrant 1-budgété, ce résultat est présenté par la Proposition 16 et illustré par la Figure 1. Notre formulation étendue peut être de taille exponentielle et sa résolution peut requérir beaucoup de temps, cependant son étude nous permet d'obtenir des propriétés structurelles sur le problème de l'arbre couvrant 1-budgété. En effet, cette formulation montre que la difficulté du problème provient de la magnitude des poids dans la contrainte de budget, de manière similaire au problème du sac-à-dos. Il en résulte que notre formulation étendue permet de résoudre le problème de l'arbre couvrant 1-budgété en temps polynomial dans le cas particulier où le nombre de poids distincts est une constante. Nous approfondirons l'exploitation de cette formulation dans le chapitre suivant dans lequel où nous étudierons la projection du polyèdre associé sur l'espace des variables originales. Dans une deuxième partie, nous présentons une formulation étendue exacte, basée sur la programmation dynamique, pour le problème de l'arbre couvrant avec plusieurs contraintes de budgets lorsque le graphe est un cactus. Dans ce cas particulier, notre formulation permet de résoudre le problème en temps pseudo-polynomial.

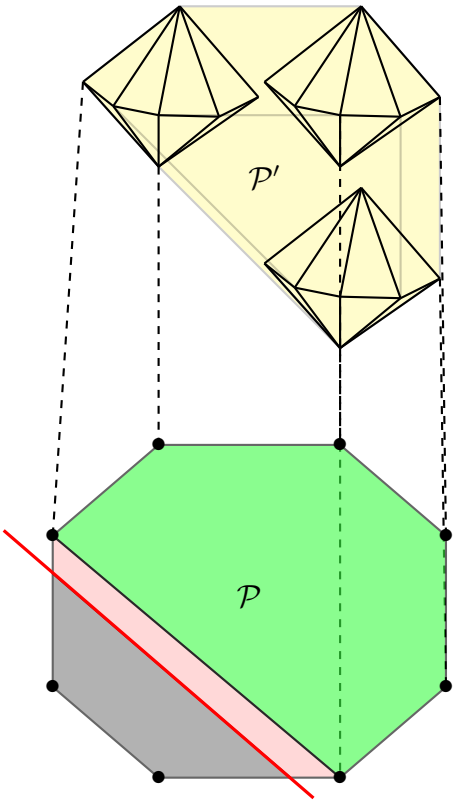


Figure 1: La projection du polyèdre $\mathcal{P}'$ associé à notre formulation étendue, ici la disjonction de trois sous-problèmes, correspond au polyèdre $\mathcal{P}$ du problème de l'arbre couvrant 1-budgété. La contrainte rouge représente la contrainte de budget et le polyèdre constitué des parties verte, rose et grise correspond au polyèdre de l'arbre couvrant

6 Inégalités valides pour le problème de l'arbre couvrant 1-budgété

Dans le sixième chapitre, nous présentons plusieurs familles d'inégalités valides pour le polyèdre de l'arbre couvrant 1-budgété ainsi que des algorithmes de séparation efficaces. Dans un premier temps, nous considérons des inégalités valides pour le problème de l'arbre couvrant. La formulation basée sur l'élimination des sous-tours décrit parfaitement le polyèdre du problème de l'arbre couvrant, les inégalités présentées dans cette section sont donc redondantes avec les contraintes des sous-tours. Cependant leurs séparations peuvent prendre du temps en pratique, afin d'être plus efficace nous présentons des algorithmes de séparation et heuristiques faisant un compromis entre la qualité des inégalités séparées et le temps d'exécution. Ces algorithmes exploitent les topologies du graphe support et de la solution courante afin de générer des inégalités de sous-tours, de coupes et/ou de multi-coupes non satisfaites. Dans un second temps, nous considérons des inégalités combinant la structure de l'arbre couvrant et du sac-à-dos. Dans la littérature, on retrouve des extensions des inégalités de couverture prenant en compte la structure de l'arbre couvrant. Bien que ces inégalités soient intéressantes en théorie, elles ne s'avèrent pas pertinentes en pratique, étant particulièrement difficiles et rares à séparer. Nous présentons des procédures

simples et rapides pour séparer des inégalités de couverture ainsi qu'une approche basée sur la résolution du problème de l'interdiction d'arbre couvrant. Cette dernière illustre la difficulté de séparer des inégalités prenant en compte la structure de l'arbre couvrant et du sac-à-dos. Dans un troisième temps, nous étudions la projection de notre formulation étendue entière présentée au chapitre précédent. Le polyèdre obtenu en projetant le polyèdre associé à la formulation étendue sur l'espace des variables initiales correspond au polyèdre du problème de l'arbre couvrant 1-budgété. L'étude du cône de projection nous permet de générer, théoriquement, les facettes du polyèdre. Cependant, la taille exponentielle du cône de projection n'est pas exploitable en pratique. Pour palier à ce problème, nous sélectionnons certains rayons du cône, associés à des inégalités valides pertinentes pour le polyèdre de l'arbre couvrant 1-budgété. Une analyse de la structure du cône de projection nous permet de caractériser des rayons aux forts potentiels parmi un nombre exponentiellement grand de générateurs. Enfin, nous proposons une procédure de séparation basée sur notre formulation étendue exacte. Cette procédure est basée sur la dualité et une formulation constituée d'une disjonction d'un nombre restreint de polyèdres. Cependant, cette procédure peut prendre du temps en pratique, une perspective d'amélioration serait de pouvoir l'incorporer dans un algorithme de coupes et branchements.

7 Algorithme de coupes et branchements pour le problème de l'arbre couvrant 1-budgété et résultats expérimentaux

Dans le septième chapitre, nous proposons un algorithme efficace de coupes et branchements pour résoudre le problème de l'arbre couvrant 1-budgété. Cet algorithme exploite les résultats des chapitres précédents. Dans un premier temps, nous présentons toutes les procédures effectuées au cours de cet algorithme; une méthode de recherche locale, un pré-traitement, une stratégie de génération d'inégalités valides à la racine ainsi qu'un algorithme de séparation. La méthode de recherche locale permet d'améliorer la solution 2-approchée obtenue en résolvant la relaxation Lagrangienne. La solution obtenue à l'issu de cette procédure est fournie au solveur en tant que MIP Start. Cette dernière permet à notre algorithme de commencer avec une solution réalisable très proche d'une solution optimale. Le pré-traitement permet de réduire la taille du problème en supprimant des arêtes ou en fixant la valeur de certaines variables de décision. Ces stratégies exploitent les propriétés de l'arbre couvrant tout en considérant les valeurs des poids. La stratégie de génération d'inégalités valides à la racine permet d'améliorer la qualité de la borne inférieure avant de commencer à brancher. En exploitant la géométrie de la relaxation Lagrangienne, calculée plus tôt, nous générons plusieurs inégalités de sous-tours, couvertures et inégalités associées à des rayons du cône de projection issu de notre formulation étendue. La combinaison de cette étape avec celle du MIP Start contribue l'obtention d'un écart très faible entre la borne inférieure et supérieure dès la résolution du sous-problème à la racine. L'algorithme de séparation reprend et combine les procédures présentées au chapitre précédent. L'efficacité de chacune de ces routines est illustrée par des tests expérimentaux intermédiaires. A cet effet, nous considérons 3×2 types d'instances, contenant des graphes de différentes densités ainsi que plusieurs distributions de coûts et poids. Les résultats montrent que chaque étape permet d'améliorer le temps de résolution. Dans un

second temps, nous présentons l'Algorithme 28, combinant toutes procédures. L'efficacité de notre algorithme est illustrée dans une troisième partie dans laquelle nous le comparons avec des algorithmes de branchements basés sur la formulation d'élimination des sous-tours ainsi que la formulation Miller-Tucker-Zemlin. Ces tests expérimentaux s'effectuent sur les mêmes instances que pour les tests intermédiaires. Les résultats montrent que notre algorithme est le plus efficace sur quasiment toutes les instances, illustrant également sa robustesse. Ce dernier permet également de résoudre des instances de tailles plus grandes que ses deux concurrents et garantit une solution réalisable proche de l'optimum grace au MIP Start. Enfin, dans une dernière partie nous donnons des statistiques supplémentaires sur la phase de séparation des différents algorithmes, illustrant l'efficacité de nos procédures. Nous menons également une étude comparant notre algorithme avec des variantes de celui-ci sur des instances avec des topologies différentes. Les résultats montrent qu'en exploitant la topologie des instances, on peut affiner notre algorithme afin d'obtenir des résultats encore meilleurs.

Le tableau 2 illustre la qualité de la solution fournie en tant que MIP Start au solveur ainsi que la rapidité des algorithmes de résolution de la relaxation Lagrangienne et de la méthode de rechercge locale.

Nombre de sommets des instances	Ecart entre la solution approchée calculée et une soluion optimale	Temps de calcul de la solution approchée
1000	0.0200	0.074
1000	0.0378	0.223
1000	0.0223	0.084
1000	0.0067	0.035
1000	0.0112	0.034
1500	0.0236	0.311
1500	0.0044	0.143
1500	0.0148	0.320
1500	0.0250	0.243
1500	0.0074	0.159

Table 2: Comparaison entre la valeur d'une solution approchée et celle d'une solution optimale

Le tableau 3 montre la dominance de notre algorithme par rapport aux algorithmes de la littérature sur les instances où le graphe est une grille et où la distribution entre les coûts et les poids est conflictuelle.

Nombre de sommets des instances	Temps de résolution (s) formulation des sous-tours	Temps de résolution (s) Algorithme 28	Temps de résolution (s) formulation Miller-Tucker-Zemlin
100	0.535	**0.035**	29.05
100	0.074	**0.037**	0.228
100	0.132	**0.058**	0.507
100	0.166	**0.050**	0.533
100	0.154	**0.036**	0.805
500	3.332	**0.471**	51.897
500	3.361	**0.312**	3600.000
500	10.109	**0.499**	78.907
500	2.968	**0.391**	89.672
500	3.101	**0.346**	274.296
1000	98.478	**3.527**	3600.000
1000	164.094	**2.996**	1121.181
1000	67.403	**0.571**	2866.763
1000	59.941	**0.641**	3600.000
1000	130.483	**5.863**	3600.000
1500	430.087	**6.274**	3600.000
1500	845.311	**25.469**	3600.000
1500	590.696	**70.491**	3600.000
1500	402.476	**128.777**	3600.000
1500	352.690	**33.872**	3600.000
2000	2845.486	**21.105**	3600.000
2000	3600.000	**1234.588**	3600.000
2000	3600.000	**670.009**	3600.000
2000	2387.236	**62.544**	3600.000
2000	3600.000	**13.888**	3600.000
2500	3600.000	**136.787**	3600.000
2500	3600.000	**3600.000**	3600.000
2500	3600.000	**67.084**	3600.000
2500	3600.000	**3578.085**	3600.000
2500	3600.000	**484.597**	3600.000
3000	3600.000	**3600.000**	3600.000
3000	3600.000	**3600.000**	3600.000
3000	3600.000	**1214.479**	3600.000
3000	3600.000	**1319.695**	3600.000
3000	3600.000	**3600.000**	3600.000

Table 3: Comparaison entre les algorithmes de branchements pour des instances où le graphe est une grille et où la distribution entre les coûts et les poids est conflictuelle

8 Algorithme de coupes et branchements pour le problème de l'arbre couvrant d-budgété et résultats expérimentaux

Dans le huitième chapitre nous présentons un algorithme de coupes et branchements pour le problème de l'arbre couvrant d-budgété. Cet algorithme est une généralisation de l'algorithme proposé au chapitre précédant pour le cas où $d = 1$. Dans une première partie, nous donnons des adaptations aux différentes procédures intermédiaires. Notre stratégie de génération d'inégalités de sous-tours à la racine est modifiée afin d'exploiter la géométrie de la relaxation Lagrangienne dans le cas où $d \geq 2$. A cet effet, nous présentons une procédure basée sur une recherche dichotomique multidimensionnelle ainsi qu'un algorithme de sous-gradient. Nous présentons une adaptation à l'algorithme de séparation du chapitre précédent, exploitant davantage les conflits apportés par les différentes contraintes de budgets. Lors de cette routine, nous résolvons le problème de faisabilité de plusieurs sous-problème d'arbre couvrant d-budgété afin de générer des inégaliés violées pertinentes. L'augmentation du nombre de contraintes de budget vient modifier la structure du problème et les autres routines (pré-traitement, calcul de la relaxation Lagrangienne et MIP Start) ne semblent plus efficaces dès lors que $d \geq 2$. De plus, notre formulation étendue, présentée au Chapitre 5 ne se généralise pas au sac-à-dos multidimensionnel. Dans une deuxième partie, nous présentons l'Algorithme 37, notre algorithme de coupes et branchements pour résoudre le problème de l'arbre couvrant d-budgété. Dans une troisième partie, nous présentons des résultats expérimentaux en considérant des instances avec 3 topologies différentes tout en faisant varier le nombre de contraintes de budget entre $d = 2, 5$ et 10. Nous comparons notre algorihtme avec des adaptations des algorithmes de branchements de la formulation des sous-tours et de la formulation Miller-Tucker-Zemlin. Les résultats montrent que notre algorithme est le plus efficace dans quasiment toutes les instances. Enfin, nous donnons également des statistiques supplémentaires sur la phase de séparation des différents algorithmes.

Le tableau 4 montre la dominance de notre algorithme par rapport aux algorithmes de la littérature sur les instances où le graphe est une grille et où $d = 5$.

Nombre de sommets des instances	Temps de résolution (s) formulation des sous-tours	Temps de résolution (s) Algorithme 37	Temps de résolution (s) formulation Miller-Tucker-Zemlin
100	**12.049**	12.884	786.616
100	59.112	**17.534**	3600.000
100	153.994	**15.357**	341.355
100	31.491	**9.017**	1253.631
100	39.876	**12.571**	71.500
200	1261.865	**109.580**	3600.000
200	3600.000	**229.230**	1800.112
200	2997.634	**704.211**	3600.000
200	1074.119	**106.008**	3600.000
200	657.371	**53.813**	1427.989
300	3600.000	**1521.693**	3600.000
300	3222.526	**260.352**	3600.000
300	3600.000	**275.055**	3600.000
300	3600.000	**189.412**	3600.000
300	3600.000	**205.920**	3600.000
400	3600.000	**915.761**	3600.000
400	3600.000	**323.467**	3600.000
400	3600.000	**434.521**	3600.000
400	3600.000	**1042.158**	3600.000
400	3600.000	**371.759**	3600.000
500	3600.000	**671.609**	3600.000
500	3600.000	**950.808**	3600.000
500	3600.000	**279.146**	3600.000
500	3600.000	**518.824**	3600.000
500	3600.000	**754.515**	3600.000
600	3600.000	**3600.000**	3600.000
600	3600.000	**1228.054**	3600.000
600	3600.000	**913.463**	3600.000
600	3600.000	**3600.000**	3600.000
600	3600.000	**3600.000**	3600.000

Table 4: Comparaison entre les algorithmes de branchements pour des instances où le graphe est une grille et où $d = 5$

9 Conclusion et travaux futurs

Dans cette thèse, nous présentons plusieurs résultats théoriques ainsi que des algorithmes exactes efficaces pour le problème de l'arbre couvrant d-budgété. Ces résultats peuvent être utilisés tels quels ou être étendus afin de résoudre de nombreuses applications.

Notre travail a également conduit à de nombreuses questions ouvertes qui seraient intéressantes à considérer dans le cadre de travaux futurs. Notre formulation étendue

exacte est sujet à de nombreuses pistes prometteuses. La taille exponentielle de celle-ci conduit chaque procédure à être chronophage. Parmi ces routines, la génération de rayons plus complexes associés à des coupes plus profondes serait particulièrement pertinente, ainsi que la diminution du temps d'exécution nécessaire pour générer ces rayons afin de pouvoir les séparer dans tous les noueds de l'arbre de branchement et non uniquement à la racine. Une autre approche intéressante consiste à concevoir une bonne approximation de l'enveloppe convexe basée sur une formulation étendue de plus petite taille afin de l'exploiter dans un algorithme de séparation basé sur la dualité.

L'ajout de contraintes budgétaires supplémentaires complique la résolution du problème. Nous avons adapté notre algorithme de coupes et branchements du cas $d = 1$ au cas où $d \geq 2$. Afin d'améliorer notre algorihme, il serait intéressant d'explorer davantage les conflits entre les différentes contraintes de budget. Une autre piste constituerait à agréger efficacement ces contraintes en une unique inégalité afin d'exploiter nos techniques pour le cas où $d = 1$.

De plus, nous proposons une formulation entière pour le problème de l'arbre couvrant d-budgété lorsque le graphe est un cactus. Dans le cadre de travaux futurs, il serait intéressant d'étendre ces résultats à tout graphe en fonction de leur largeur arborescente.

Enfin, il serait également intéressant d'étendre nos résultats à d'autres problèmes de réseaux soumis à une ou plusieurs contraintes de sac-à-dos et d'exploiter cette relation entre un problème bien structuré et un problème volatil afin d'obtenir des algorithmes efficaces.

Contents

1 Introduction **17**
- 1.1 Related works . . . 18
  - 1.1.1 Multiobjective spanning tree problem . . . 18
  - 1.1.2 Minimum spanning tree problem with a single budget constraint ($d = 1$) . . . 19
  - 1.1.3 Minimum spanning tree problem with multiple budget constraints ($d \geq 2$) . . . 20
- 1.2 Our works . . . 20
- 1.3 Notations . . . 22

2 Basic notions **24**
- 2.1 Combinatorial optimization . . . 24
- 2.2 Computational complexity . . . 24
- 2.3 Polyhedral approach . . . 25
- 2.4 Combinatorial optimization and linear programming . . . 27
- 2.5 Extended linear formulations . . . 29
- 2.6 Disjunctive programming . . . 30
- 2.7 Graph theory . . . 30
- 2.8 Matroids . . . 31

3 The minimum d-budgeted spanning tree problem and polyhedra **33**
- 3.1 Formulations . . . 33
  - 3.1.1 Subtour elimination formulation . . . 34
  - 3.1.2 Cut formulation . . . 34
  - 3.1.3 Miller-Tucker-Zemlin formulation . . . 35
- 3.2 Polyhedral analysis . . . 36
  - 3.2.1 Dimension . . . 36
  - 3.2.2 Facial aspect . . . 41
- 3.3 Special cases of the d-budgeted spanning tree problem . . . 43

4 Linear and Lagrangian relaxations for the minimum 1-budgeted spanning tree problem **45**
- 4.1 Linear programming relaxation . . . 45
- 4.2 Lagrangian relaxation . . . 49
  - 4.2.1 Solving efficiently the Lagrangian relaxation . . . 51
  - 4.2.2 Special case of the Lagrangian relaxation . . . 61

5 Extended formulations for the 1-budgeted spanning tree problem 66
5.1 Extended formulation based on matroid intersection 66
5.2 Pseudo-polynomial size extended formulation for cactus graphs 74

6 Valid inequalities for the 1-budgeted spanning tree problem 79
6.1 Separating efficiently spanning tree's inequalities 80
6.2 Separating efficiently cover inequalities 84
6.3 Valid inequalities obtained via projection 87
6.3.1 A priori valid inequalities 87
6.3.2 Cutting plane approach 93

7 Branch-and-Cut for the minimum 1-budgeted spanning tree problem and computational experiments 97
7.1 Problem classes 97
7.2 Benchmark algorithms 98
7.3 Preprocessing and probing procedures 100
7.3.1 Local search method 100
7.3.2 Probing 102
7.3.3 Cut generation 105
7.4 Cutting plane algorithm 107
7.5 Algorithm for the minimum 1-budgeted spanning tree problem 110
7.6 Computational experiments 112
7.6.1 Conflicted instances 112
7.6.2 Uncorrelated instances 116
7.6.3 Separation statistics and variants 118

8 Branch-and-Cut for the minimum d-budgeted spanning tree problem and computational experiments 125
8.1 Preprocessing 126
8.2 Cutting plane algorithms and heuristics 129
8.2.1 Separation of spanning tree's valid inequalities 129
8.2.2 Separation of cover inequalities 130
8.3 Algorithm for the minimum d-budgeted spanning tree problem 133
8.4 Computational experiments 133

9 Conclusion and future works 145

10 Appendices 146

Chapter 1

Introduction

The minimum spanning tree problem (MSTP) is one of the most well known problems of combinatorial optimization. The importance and the popularity of the MSTP stems in part from the simplicity and the efficiency of its algorithms (Kruskal, Prim and Borůvka algorithms). It has direct applications in the design of communication, power, and transportation networks [32, 17, 46, 16]. The MSTP provides solutions to other problems to which it applies indirectly such as network reliability, clustering, and classification problems [54, 81]. MSTP algorithms are also used as subroutines in several exact and approximate algorithms for the traveling salesman problem, the multiterminal flow problem, etc [38, 39, 31].

All the methods for the solution of the MSTP are based on the greedy algorithm. This algorithmic paradigm can be applied to various other problems and it is studied more generally in the theory of matroids [58].

Most real-life optimization problems involve finding a solution trading off many conflicting objectives [26, 77, 29, 20]. Multiobjective optimization is a rich area of study in Operations Research, Computer Science and Economics. A variety of models have been used to formulate such problems including Goal Programming [43], Pareto-optimality [47, 15], and the $\epsilon-$constraint method [41]. We adopt the latter approach and cast one of the objectives as the objective function, and the others as budget constraints. The resulting class of optimization problems is called budgeted optimization problems. More precisely, the minimum d-budgeted problem is defined as follows:

Definition 1. Consider a connected graph $G = (V, E)$ such that every edge $e \in E$ has a cost $c_e \in \mathbb{R}^+$ and weights $w_e^i \in \mathbb{N}, i = 1, \ldots, d,\ d \in \mathbb{N}$, and let $B^i \in \mathbb{N}, i = 1, \ldots, d$ denote budgets. The Minimum d-Budgeted Spanning Tree Problem (MdBSTP) consists in finding a spanning tree $T = (V, F)$ such that $\sum_{e \in F} w_e^i \leq B^i,\ \forall i = 1, \ldots, d$ and $\sum_{e \in F} c_e$ is minimum.

The MdBSTP lies at the intersection of the spanning tree and the multi-knapsack problems. It is weakly NP-hard if d is a constant and strongly NP-hard otherwise.

1.1 Related works

1.1.1 Multiobjective spanning tree problem

In multiobjective optimization, there are more than one objective functions and there is no single solution simultaneously optimizing all the objective functions. A key concept in this field is the Pareto optimality which replaces the notion of optimality. Pareto optimal solutions are solutions that can not be improved in one objective function without deteriorating at least one of the other objective function. More formally, a spanning tree $T = (V, F)$ is Pareto optimal if there is no other spanning tree $T' = (V, F')$ such that $\sum_{e \in F'} c_e^i \leq \sum_{e \in F} c_e^i,\ i = 1, \ldots, d$, with at least one strict inequality.

The $\epsilon-$ constraint method is a popular approach in multiobjective optimization. By varying the budget bounds B^i, the efficient Pareto optimal solutions can be obtained. Figure 1.1 gives an illustration in the case $d = 1$.

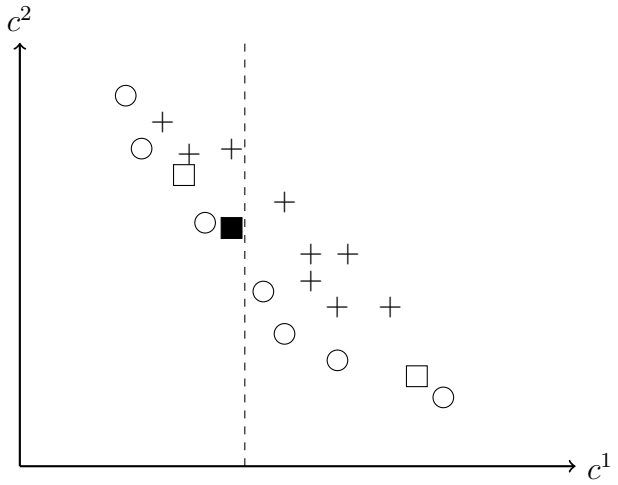


Figure 1.1: The optimal solution of the ϵ-constraint method on a set of solutions. The pareto front is given by all the circles and squares, where a circle (respectively a square) represents a supported (respectively a non-supported) solution. The optimal solution of the ϵ-constraint method for a given value of B^1 is given in black.

Due to the exponential size of the set of Pareto optimal solutions, enumerating all of them is time consuming. The two phase method is a general method to enumerate the Pareto optimal solutions. It was first introduced by Ulungu and Teghem [75] and adapted by Ramos et al. [64] to the spanning tree problem. In phase I, supported Pareto optimal solutions are computed. This step exploits a fundamental geometric property that the latter are optimal solutions of weighted sums of the objective functions c^i. In Phase II, the remaining Pareto optimal solutions are formed by enumeration methods. Sourd and Spanjaard [72] present a multiobjective extension of the Branch-and-Bound algorithm to enumerate all the Pareto optimal solutions in the case $d = 2$. The performance of the algorithm is enhanced by implementing a generalization of the classical cut and cycle rules.

Stidsen et al. [73] propose a Branch-and-Bound algorithm to solve biobjective mixed integer programs where the integer variables are binary, and one of the two biojective has

only integer variables. A survey of the state of the art multiobjective Branch-and-bound is given in [63].

In the particular case $d = 2$, Aggarwal et al. [1] combined the geometry of the criteria space (Figure 1.1) and devise a Branch-and-Bound to solve optimally the budgeted problem.

Andersen et al. [5] propose heuristics based on neighborhood and adjacent search to quickly compute an approximation of the set of Pareto optimal solutions.

1.1.2 Minimum spanning tree problem with a single budget constraint ($d = 1$)

Yamada et al. [82] propose a Branch-and-Bound algorithm to solve the maximization version of the problem and propose a heuristic, based on Lagrangian relaxation and local search to accelerate the computation. The algorithm is tested on plane and complete graphs with correlated (easy instances) and uncorrelated (difficult instances) c^1 and c^2 functions. The numerical experimentation shows that the algorithm is able to solve easy instances with up to 1000 nodes but difficult instances with only up to 100 nodes. Unfortunately, no implementation details are given in the paper. The authors suggested that the investigation of a polyhedral approach may improve the running time.

Agra et al. [2] investigate strengthening the Miller-Tucker-Zemlin (MTZ) formulation [53] by adding valid inequalities. The latter correspond to a generalization of the cover inequalities that combine the information from the spanning tree and knapsack polytopes. The authors claim that interesting results are obtained by incorporating these inequalities in a Branch-and-Cut algorithm. However, we carried an in-depth experimental study and showed the strengthened formulation has a comparable performance to the standard MTZ formulation. Notice also that their formulation requires strictly positive weights to be valid. Agra et al. extended this family of inequalities to problems where solutions correspond to basis of a matroid subject to a knapsack constraint [3]. They also discuss about lifting procedures enhancing the inequalities. Although those inequalities seem theoretically relevant, they appeared to be not very useful in practice on our instances.

Hong et al. [40] propose an exact pseudo-polynomial algorithm to solve the problem based on a two-variable extension of the matrix-tree theorem. This technique could be generalized to a constant number of $d \geq 2$ of budget constraints.

Ravi and Goemans [65] give an approximation algorithm for the problem based on the Lagrangian relaxation technique. The algorithm exploits the combinatorial property that two adjacent spanning trees on the spanning tree polytope differ by exactly two edges. The algorithm can be implemented in parallel by using the technique of Megiddo [51]. Hassin and Levin [37] provide a faster approximation algorithm by combining the idea of Goemans and Ravi [65] with matroid intersection algorithm. An approximate solution is computed by solving a polynomial number of matroid intersection problems instead of solving a budgeted spanning tree problem. Several works [66, 67, 60] combine Lagrangian heuristics with local-search techniques to efficiently compute a solution as close as possible to an optimal solution. Although those algorithms do not improve the guaranty of performance given by the Lagrangian heuristics, experimental results, given by the authors, show that those algorithms appear to be effective in practice.

1.1.3 Minimum spanning tree problem with multiple budget constraints ($d \geq 2$)

Shogan [71] presents an exact algorithm for the minimum spanning tree with several budget constraints and flow requirements. Notice that the MdBSTP is a special case of the latter problem in which every demand is equal to 1. Shogan proposes a Branch-and-Bound algorithm where the lower and upper bounds are computed via Lagrangian relaxation. Computational experiments are presented, with instances up to 50 nodes and 5 budget constraints.

MdBSTP generalizes several well-known combinatorial optimization problems such as the *degree-constrained minimum spanning tree* introduced by Narula and Ho [56]. The problem consists in finding a spanning tree such that the degree of every vertex $v \in V$ is at most $d_v \in \mathbb{N}$. The problem can be formulated as a $|V|$BSTP, with a degree constraint for every vertex. Goemans [30], Andrade et al. [6] give approximation algorithms for the minimum bounded spanning tree based on a Lagrangian relaxation. Kumar and Singamsetty [45] consider a variant where the spanning tree is subject to degree constraints and budget constraints. The authors also give an exact algorithm based on pattern recognition.

Samer and Urrutia [68] study the *spanning tree problem with several conflict constraints*, a special case of the dBSTP where a budget constraint is associated with each conflict constraint. The authors introduce several valid inequalities for the problem and propose a Branch-and-Cut algorithm. They conduct computational experiments with instances up to 300 nodes and 14985 conflict constraints.

Olver and Zenklusen [57] considered the *chain-constrained spanning tree problem* with several budget constraints on sets $\emptyset \subsetneq S_1 \subsetneq \cdots \subsetneq S_d \subset E$. They propose an approximation algorithm based on matroid intersection.

Other budgeted network problems are studied in the literature such as the arborescence [35], the shortest path [28], the matching [10] and assignment problems [80].

1.2 Our works

The difficulty of the dBSTP stems from the arbitrary nature of the cost and weight values. With the exception of well-known classes of valid inequalities, such as the cover inequalities, it is not easy identify other classes. By solving spanning tree interdiction subproblems, we managed to generate cover inequalities that appear to be more efficient than the idea of Agra et al. [2]. However, these families of inequalities remain weak in some cases. The challenge is to identify strong valid inequalities that exploit both the combinatorial structure of the spanning tree and knapsack polytope. Building on the work of Hassin and Levin [37] and Balas [8], we propose an exact extended formulation of the problem, in the case $d = 1$, mixing disjunctive programming and matroids intersection results. In theory, it is possible to obtain all the facets characterizing the convex hull via projection. However, the number of generators is exponentially large. By carefully selecting rays of the projection cone that exploit the spanning tree and knapsack polytopes, we generate deep cuts. The Lagrangian relaxation technique is used indirectly to generate these "good" rays and compute the tightest right-hand side of the induced valid inequalities.

We imbed the projection technique in the framework of a Branch-and-Cut algorithm. In order to improve the running time, we propose a fast algorithm to solve the Lagrangian relaxation problem, and efficient separation routines. We conducted a through experimental study and we generated different classes of instances. We show that our algorithm outperforms existing algorithms and benchmark Branch-and-cut algorithm implemented on Gurobi, one of the best solver. In order to find effectively an optimal solution, the later adds cuts and runs sophisticated branching strategies. Gurobi also has heuristics that can aid finding a good initial solution. Furthermore, it includes a sophisticated preprocessing system. These results led to a paper submitted to the *INFORMS Journal on Computing*.

Adding more budget constraints makes the problem hard to solve. We adapt and extend our algorithm to handle more budget constraints. Our algorithm also dominates, in this case, the default Branch-and-Cut algorithms.

The thesis is organized as follows. In Chapter 2, we give basic notions about combinatorial optimization, polyhedral theory and key definitions for the entire document. In Chapter 3, we give several integer formulations for the MdBSTP and investigate its related polytope. We also study a special case of the problem. In Chapter 4, 5, 6 and 7 we focus on the special case where $d = 1$. In Chapter 4, we investigate the linear and Lagrangian relaxation of the problem. We propose an efficient algorithm to solve the Lagrangian relaxation. In Chapter 5 we present an exact extended formulation for the M1BSTP. We also give an exact pseudo-polynomial size extended formulation for the d-budgeted case when the graph is a cactus graph. In Chapter 6, we give efficient separation algorithms for spanning tree and cover inequalities. We then discuss the projection of our extended formulation to the original variable space. This formulation is exploited in order to generate several valid inequalities and cutting plane algorithm for the problem. In Chapter 7 we devise a Branch-and-Cut algorithm that takes advantage of the results of the previous chapters. We present computation experiments in order to show its efficiency. In Chapter 8 we generalize every ideas of the previous chapters to devise a Branch-and-Cut algorithm for the minimum d-budgeted spanning tree problem. We also give computation experiments.

1.3 Notations

In this section, the reader can find the signification of several notations that will be used in the entire document. Those notations will also be defined later in the document.

$d \in \mathbb{N}$	Number of budget constraints considered in the problem,
$B^i \in \mathbb{N}$	Budgets of the $i = 1, \ldots, d$ resource constraints,
$G = (V, E)$	Undirected simple connected graph,
$c : E \to \mathbb{R}^+$	Cost function,
$w^i : E \to \mathbb{N}$	Weight functions related to resource $i = 1, \ldots, d$, Without loss of generality, all weights and costs can be assumed to be positive (by adding a fixed positive value to the weights or cost to all edges, and adapting the related budgets, the optimal solutions remain unchanged as all solutions have the same cardinality),
MdBSTP	Minimum d-Budgeted Spanning Tree Problem,
STP	Spanning Tree Problem,
KP	Knapsack Problem,
MTZ	Miller-Tucker-Zemlin,
P_{ST}	Spanning tree polytope,
P_{sub}	Subtour formulation polytope,
$conv(S)$	Convex hull of S, where S is a set of solutions,
$\mathcal{H}_{ST}$	Set of incidence vectors of the STP,
$\mathcal{H}$	Set of incidence vectors of the dBSTP,
$\mathcal{P}$	$\mathcal{P} = conv(\mathcal{H})$, polytope of the dBSTP,
$\mathcal{P}_i$	$\mathcal{P}_i = conv(\{x \in \{0,1\}^{\lvert E\rvert} \mid x \in P_{sub}, \sum_{e \in E} w_e^i x_e \leq B^i\})$,
$d(P)$	Dimension of the polytope P,
E_x^0	Forbidden edge set, $E_x^0 = \{e \in E \mid x_e = 0,\ \forall x \in \mathcal{H}\}$,
E_x^1	Necessary edge set,$E_x^1 = \{e \in E \mid x_e = 1,\ \forall x \in \mathcal{H}\}$,
q	Number of induced subgraphs of G separated by articulation vertices,
$G^* = (V, E^*)$	Subgraph of $G = (V, E)$ where $E^* = \{e \in E \mid x_e^* > 0\}$ and $x \in [0; 1]^{\lvert E\rvert}$,
branches	Edge set removed by Algorithm 3,
$L(z),\ z \geq 0$	Piece-wise linear function associated with the Lagrangian relaxation,
z^*	$z-$coordinate of the maximum of the Lagrangian function $L(z)$,
$T^< = (V, F^<)$	Feasible solution obtained by rounding an optimal solution of the linear relaxation,
$T^> = (V, F^>)$	Infeasible solution obtained by rounding an optimal solution of the linear relaxation,
E^0	$E_0 = \{e \in E \mid c_e = 0\}$,
E^1	$E_1 = \{e \in E \mid c_e = 1\}$,

H^0	Edge set of a forest minimizing lexicographically $(-\|H^0\|, \sum_{e\in H^0} w_e)$ in the subgraph $G^0 = (V, E^0)$,
H^1	Edge set of the forest minimizing lexicographically $(-\|H^1\|, \sum_{e\in H^1} w_e)$ in the subgraph $G^1 = (V, E^1)$,
k	Number of distinct weight values,
P_h	$P_h = \{y \in \mathbb{R}^{\|E\|} \mid Ay^h \leq b^h\}, h \in \mathbb{N}^k$, matroid intersection polytope of the graphic and partition matroids related to h,
Q^*	Set of vectors $h \in \mathbb{N}^k$ satisfying Constraints 1, 2 and 3 such that the related polytope P_h is feasible,
$\mathcal{P}_h$	$\mathcal{P}_h = \{x \in P_h \mid \sum_{e\in E} x_e = \|V\| - 1\}, h \in Q^*$,
$\mathcal{P}'$	Polytope of the extended formulation (5.18)-(5.22) based on matroid intersection and disjunction,
$\mathcal{C}$	Polyhedral cone of the projection of $\mathcal{P}'$ onto the x-space,
r	Ray of the polyhedral cone $\mathcal{C}$, $r = (r_{card}, r_E, r_{y_0}, r^{h_1}, r^{h_1}_{y_0}, \ldots, , r^{h_{\|Q^*\|}}, r^{h_{\|Q^*\|}}_{y_0})$,
$G_{ext} = (V_{ext}, A_{ext})$	Extended graph based on dynamic programming,
K, K'	Upper bounds on the total number of valid inequalities added during the cutting phase,
$\alpha, \beta^h, h \in Q^*, \gamma$	Dual variables,
$Grid$	Instances with a grid graph structure,
$Geom$	Instances with a random geometric structure inspired by [42],
$Rand$	Instances with a random dense structure,
I	Conflicted instances,
R	Uncorrelated instances,
RT_X	Total resolution time of strategy X, given in seconds,
ST_X	Total separation time of strategy X, given in seconds,
NC_X	Total number of valid inequalities separated, added to strategy X during the resolution,
NN_X	Total number of nodes in the branching tree of strategy X,
Gap_X	Gap between the best integer solution found and the best lower bound at the end of the resolution of strategy X, given in percent, a $+\infty$ signifies that no feasible solution has been found,
ST	Subtour formulation, Algorithm 17,
$Alg28$	Algorithm 28 for the M1BSTP,
MTZ	MTZ based formulation (3.14)-(3.20) with inequalities (7.1),
Var_1, Var_2	Variants of Algorithm 28,
$Alg37$	Algorithm 37 for the MdBSTP.

Chapter 2

Basic notions

In this chapter we present basic notions and definitions about structures and approaches that will be used in this thesis.

2.1 Combinatorial optimization

Combinatorial optimization is a subfield of operations research, situated at the intersection of computer science and applied mathematics. The objective is to find an optimal solution among a finite set of solutions. Let $E = \{e_1, \ldots, e_n\},\ n \in \mathbb{N}^*$ be a finite set. Let c_{e_i} be a cost associated with e_i, for $i = 1, \ldots, n$. Let $\mathcal{F}$ be a family of subsets of E, the cost of any $F \in \mathcal{F}$ corresponds to $c(F) = \sum_{e_i \in F} c_{e_i}$. The problem that consists in finding an element $F^* \in \mathcal{F}$ such that $c(F^*) = \max(or \min)\{c(F) \mid F \in \mathcal{F}\}$ is called a *combinatorial optimization problem.*

The term *combinatorial* refers to the finite set of solutions $\mathcal{F}$ and *optimization* as we are searching for one of the best elements. One can find an optimal solution by enumerating every solution $F \in \mathcal{F}$, computing every related $c(F)$ and selecting an element maximizing the total cost. In practice, this is not tractable as the size of $\mathcal{F}$ is exponential in most of the combinatorial optimization problems.

The interest of combinatorial optimization is closely related to real life applications. It appears in several fields where efficiency is vital or essential such as telecommunication, logistic, supply chain, scheduling or transport. Resulting from all those needs, several approaches have been developed in order to quickly solve combinatorial optimization problems. Most of them are based on graph theory or polyhedral approaches. Since the 20th century, many results, techniques and technological innovations have made these approaches very effective in order to solve real-world problems.

2.2 Computational complexity

Computational complexity theory is a field in theoretical computer science, classifying problems depending on the quantity of resources, usually time and storage, required to solve the problem. Since the formalization of Turing machine by Alan Turing, several

studies have been made in order to characterize what is a "good" algorithm. An *algorithm* is a sequence of elementary operations used to solve a given instance of a problem. An *instance* of a problem corresponds to the data and parameters in which we solve the problem. An algorithm is said to be *polynomial* if the number of elementary operations required to solve any instance of size n in the worst case is bounded by a polynomial function in n. Moreover, an algorithm is said to be *pseudo-polynomial* if the number of elementary operations required to solve any instance of size n in the worst case is bounded by a polynomial function in n and in the magnitude of the data and parameters of the input. Such algorithms have an exponential behavior in function of their input size, and are then not considered as polynomial algorithms.

A *decision problem* is a question such that the answer can be only "yes" or "no". P is the class of all decision problems that can be solved using a polynomial algorithm. NP (Nondeterministic Polynomial) is the class of all decision problems such that given an instance the "yes" answer can be verified using a polynomial algorithm. By definition, we have $P \subseteq NP$. The question "Does $P = NP$?" is still a major unsolved problem and belongs to the Millennium Prize Problems. Although there exists no proof indicating the contrary, it is very unlikely that $P = NP$. A decision problem is said to be *NP-complete* if it belongs to NP and every problem of the NP class admits a polynomial reduction to it. A *reduction* consists of transforming an instance of a given problem to an equivalent instance of another problem. A reduction is *polynomial* when the number of modifications can be bounded by a polynomial function. A NP complete problem is *weakly*-NP *complete* if there is an algorithm solving any instance of the problem in pseudo-polynomial time. Every combinatorial optimization problem is associated with a decision problem. If the related decision problem is NP-complete (respectively weakly NP-complete), then the combinatorial optimization problem is said to be *NP-hard* (respectively weakly NP-hard).

Given two functions $f(n)$ and $g(n)$, the function f is bounded by the function g using the big O notation; and we write $f(n) = O(g(n))$ if there exist constants n_0 and C such that for all $n \geq n_0, f(n) \leq Cg(n)$.

For more details on computational complexity, the reader can see [22].

2.3 Polyhedral approach

Polyhedral theory is a subfield of discrete geometry, studying the structure of convex sets of solutions described by a system of inequalities. Given a set of points $x^1, \ldots, x^m \in \mathbb{R}^n$, $n, m \in \mathbb{N}^*$, $x \in \mathbb{R}^n$ is a *convex combination* of $x^1, \ldots, x^m$ if $x = \sum_{i=1}^m \lambda_i x_i$, $\lambda_i \geq 0$ for $i = 1, \ldots, m$ and $\sum_{i=1}^m \lambda_i = 1$. The *convex hull* of a set of points $S = \{x^1, \ldots, x^m\} \in \mathbb{R}^{n \times m}$, denoted by $conv(S)$, is the set of all points $x \in \mathbb{R}^n$ that are convex combinations of $x^1, \ldots, x^m$.

A *polyhedron* is the set of solutions satisfying a system of inequalities $Ax \leq b$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. A *polytope* is a bounded polyhedron. A solution is an *extreme point* of a polyhedron P if and only if it cannot be obtained by a convex combination of at least two solutions of P.

A subset C of $\mathbb{R}^n$ is called a *cone* if $C \neq \emptyset$ and $\lambda x + \mu y \in C$ for all $x, y \in C$ and

$\lambda, \mu \in \mathbb{R}_+$. A cone C is polyhedral if there is a matrix A such that $C = \{x \in \mathbb{R}^n \mid Ax \leq 0\}$. A non zero vector $r \in C$ is an *extreme ray* of C if there do not exist linearly independent $x, y \in C$ and positive scalars λ, μ such that $r = \lambda x + \mu y$.

A polyhedron (respectively a polytope) can be described by its extreme points and extreme rays (respectively by its extreme points). Figure 2.1 illustrates the above definition.

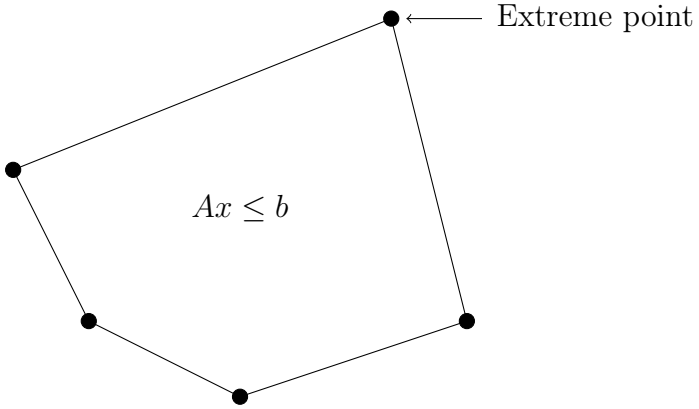


Figure 2.1: A polytope.

The points $x^1, \ldots, x^m \in \mathbb{R}^n$ are *affinely independent* if the following system $\sum_{i=1}^m \lambda_i x^i = 0, \sum_{i=1}^m \lambda_i = 0$ has a unique solution $\lambda_1 = \lambda_2 = \cdots = \lambda_m = 0$. The *dimension* of a polyhedron P is the maximum number of affinely independent solutions in P minus 1. We denote the dimension of P by $d(P)$. Moreover, if $P \subseteq \mathbb{R}^n$, then $d(P) \leq n$. If $dim(P) = n$ then P is said to be *full dimensional*. Consider a polyhedron $P = \{x \in \mathbb{R}^n | \ Ax \leq b, Bx = d\}$ where B contains all equations satisfied by all solutions of P, if $P \neq \emptyset$, then $d(P) = n - rank(B)$.

Given a polyhedron P, an inequality $ax \leq b$ is said to be *valid* for P if $ax \leq b$ for every $x \in P$, hence $P \subseteq \{x \in \mathbb{R}^n | \ ax \leq b\}$. Given a valid inequality $ax \leq b$ of P, the set of solutions $F = \{x \in P | \ ax = b\}$ is called a *face* of P induced by $ax \leq b$. A face F of P is said to be *proper* if $F \neq \emptyset$ and $F \neq P$. Moreover, a face F of P is called a *facet* of P if F is proper and $d(F) = d(P) - 1$. In order to describe a polyhedron by a linear inequality system, one needs to determine its facets. Any inequality $ax \leq b$ which does not define a facet of P ($P \neq \emptyset$) is redundant in the system describing P.

Figure 2.2 illustrates the above definitions.

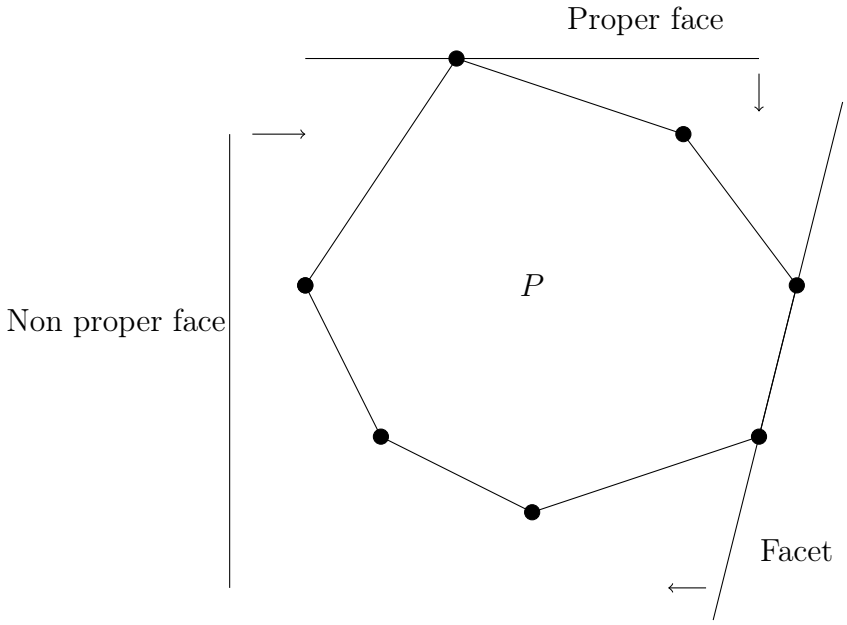


Figure 2.2: Faces and facets.

2.4 Combinatorial optimization and linear programming

Polyhedral theory appears to be very effective in order to solve combinatorial optimization problems. Any combinatorial optimization problem can be associated with a polyhedron. Given a combinatorial optimization problem with a family $\mathcal{F}$ of subsets of $E = \{e_1, \ldots, e_n\}$, we represent each solution of $\mathcal{F}$ by a 0-1 vector. Given any solution $F \in \mathcal{F}$, let x^F be the vector such that $x^F_{e_i} = 1$ if $e_i \in F$, $x^F_{e_i} = 0$ otherwise, $i = 1, \ldots, n$. x^F is called the *incidence vector* of F. The convex hull $conv(\{x^F | F \in \mathcal{F}\})$ is the polyhedron describing the associated combinatorial optimization problem. Determining an optimal solution of the combinatorial problem reduces to find $F^* \in \mathcal{F}$ such that $c(F^*) = \max(\text{or } \min)\{cx^F \mid F \in \mathcal{F}\} = \max(\text{or } \min)\{cx^F \mid x \in conv(\{x^F \mid F \in \mathcal{F}\})\}$. The problem can be reduced to the *linear program*:

$$\begin{aligned} \max \quad & cx \\ \text{s.t.} \quad & Ax \leq b, \\ & 0 \leq x \leq \mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}. \end{aligned}$$

When all the facets of the polyhedron are known, solving the combinatorial optimization problem is equivalent to solving the related linear program. In the opposite case, when the full linear description is not known, integrality constraints are required for formulating the problem:

$$\begin{aligned} \max \quad & cx \\ \text{s.t.} \quad & Ax \leq b, \\ & x \in \{0,1\}^n. \end{aligned}$$

Adding integrality constraints leads to an *integer linear problem*. The resulting problem is harder than its relaxed version.

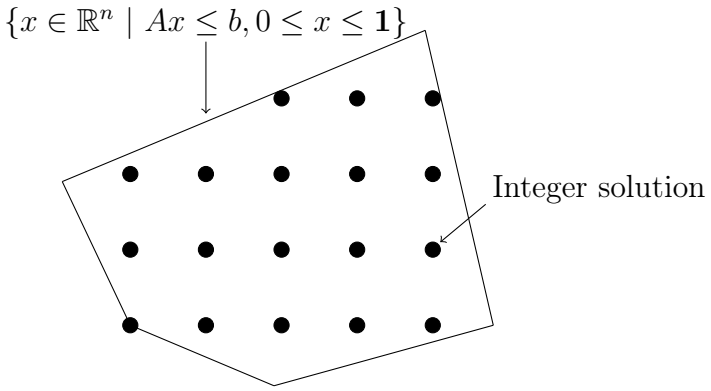


Figure 2.3: Relaxed description of an integer linear problem. Every point represents a feasible integer solution.

A *compact* formulation is a (integer) linear program such that the number of constraints and variables is bounded by a polynomial function. Any linear program with a compact formulation can be solved in polynomial time using the elipsoid method. Several combinatorial optimization problems admit a description with an exponential number of constraints. In this case, the linear program can be solved in polynomial time using a *cutting plane algorithm*. Given a polyhedron P and a point x^* of $\mathbb{R}^n$, the cutting plane method is based on the *separation problem*, consisting of verifying if $x^* \in P$, and if this is not the case, to find an inequality $ax \leq b$, valid for P and violated by x^*. A linear program with an exponential number of constraints can be solved in polynomial time if and only if the associated separation problem can be solved in polynomial time. Indeed, by alternating the resolution of the enhanced linear program and separation procedure, it is possible to solve the whole linear program in polynomial time if the separation procedure is polynomial [33]. Cutting plane algorithm can be described as follows.

Algorithm 1: A cutting plane algorithm

Data: A linear program LP and its system of inequalities $Ax \leq b$
Result: Optimal solution of LP

```
Consider a linear program LP' with a reasonable number of inequalities among
  Ax ≤ b;
do
    Solve LP' and let x* be an optimal solution;
    Solve the separation problem associated with Ax ≤ b and x*;
    if an inequality ax ≤ α is violated then
        Add ax ≤ α to LP';
    end
while inequalities can be added to LP';
```

An integer linear program cannot be solved by considering the related relaxed problem, several fractional variables may arise, the obtained solution leads to an upper bound (lower for a minimization problem) on the optimal solution. In most of the cases, rounding the fractional solution to an integer solution is not enough in order to find the optimal integer solution. The *Branch-and-Bound* approach explore several subproblems in order to converge to the optimal solution. This procedure builds a resolution tree such that every node corresponds to a subproblem. The initial problem, situated at the root is $LP_0 = \max\{cx \mid Ax \leq b, 0 \leq x \leq 1\}$. The linear problem LP_0 is solved, let x_0^* be the optimal solution of LP_0. If x_0^* is integral, then it is optimal, the algorithm stops. Otherwise, if x_0^* is fractional, the algorithm applies a *branching* procedure. A fractional variable x^1 is selected and two nodes P_1 and P_2 are added to the tree. The vertex P_1 (respectively P_2) is associated with the linear subproblem $LP_1 = \max\{cx \mid Ax \leq b, x^1 = 0, 0 \leq x \leq 1\}$ (respectively $LP_2 = \max\{cx \mid Ax \leq b, x^1 = 1, 0 \leq x \leq 1\}$). The linear problem LP_1 (LP_2) is then solved. If the optimal solution of LP_1 is integral, the node P_1 is *pruned* and becomes a leaf of the tree. In the other case, a fractional variable is selected and two subproblems of LP_1 are created. At every step, the procedure chooses a subproblem that is not a leaf with an upper bound (lower for a minimization problem) higher than the best integer solution found (if any) until every node is pruned or has a bound lower than the best feasible solution found.

A *Branch-and-Cut* algorithm combines the exploration of a Branch-and-Bound with cutting plane procedures on the nodes of the tree. This enable to compute better bounds and then reduce the computational time of the problem.

For more details on polyhedral theory, the reader can see [70, 79].

2.5 Extended linear formulations

Several combinatorial optimization problems can only be described by an exponential size formulation in the incidence vectors space. Additional variables allows one to add structure in order to find formulations with more properties like compact or integer description. Consider a polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ for a given combinatorial optimization problem. An *extended formulation*, considering further variables $y \in \mathbb{R}^p$, for the problem

is given by $Q = \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^p \mid Bx + Cy \leq d\}$.

Extended formulations have interest in both theory and practice. A better description of the convex hull of the incidence vectors in the original variable space can be obtained by projecting Q onto the x-space. We denote by $Proj_x(Q) = \{x \in \mathbb{R}^n \mid \exists (x, y) \in Q\}$ the projection of Q onto the x-space. The *projection cone* of Q is given by $\mathcal{C} = \{v \in \mathbb{R}^m \mid vC = 0, v \geq 0\}$, where m is the number of rows of B and C. The projection of Q can be obtained by exploiting the projection cone $\mathcal{C}$ as $Proj_x(Q) = \{x \in \mathbb{R}^n \mid (vB)x \leq vd, v \in extr(\mathcal{C})\}$. Where $extr(\mathcal{C})$ is the set of extreme rays of $\mathcal{C}$. Each extreme ray of $\mathcal{C}$ satisfies with equality $p - 1$ linearly independent inequalities of the system $vC = 0, v \geq 0$. For more details on extended formulations and projections, the reader can see [19].

2.6 Disjunctive programming

Disjunctive programming deals with optimization problems on disjunctive sets of solutions, that is disjunctions of polyhedra. Although every polyhedron is convex, the union of several polyhedra may not be convex. Let $P_i = \{x \in \mathbb{R}^n \mid A^i x \leq b^i\},\ i = 1, \ldots, q$ be a finite number of non-empty polyhedra. Balas has introduced a formulation related to the disjunction of all polyhedra $P_h,\ h \in \{1, \ldots, q\}$. In this extended formulation, new variables y^h and $y_0^h, h \in \{1, \ldots, q\}$ are introduced. Variables y^h are copies of the original variables x for each polyhedron P_h. Variables y_0^h take value 1 if the solution belongs to the polyhedron P_h, 0 otherwise. The description is given below:

$$\min\ c^T x \tag{2.1}$$

$$x - \sum_{h=1}^{q} y^h = 0 \tag{2.2}$$

$$A^h y^h - b^h y_0^h \leq 0 \qquad \forall h = 1, \ldots, q \tag{2.3}$$

$$\sum_{h=1}^{q} y_0^h = 1 \tag{2.4}$$

$$y_0^h \geq 0 \qquad \forall h = 1, \ldots, q \tag{2.5}$$

The extreme points of the polyhedron (2.2)-(2.5) coincide with extreme points of every polyhedron $P_i = \{x \in \mathbb{R}^n \mid A^i x \leq b^i\},\ i = 1, \ldots, q$. For more details on disjunctive programming, the reader can see [8].

2.7 Graph theory

Graph theory is a field of discrete mathematics and Computer science, dealing with structures describing pairwise relations between elements. An *undirected graph* is a pair $G = (V, E)$ where V is a set of elements called *vertices* and E is a set of pairs of vertices called *edges*. If $\{u, v\} \in E$ is an edge with endnodes u and v, we will also write uv to denote the edge. A *directed graph* is a pair $G = (V, A)$ where V is a set of elements called vertices and

A is a set of ordered pairs of vertices called *arcs*. If $(u, v) \in A$ is an arc from u to v, we will also write uv to denote the arc. A graph G is said to be *simple* if it does not have more than one edge between any couple of vertices, and it has no edge that starts and ends at the same vertex. Otherwise, G is called a *multigraph*.

A *subgraph* $G' = (S, H)$ of a graph $G = (V, E)$ is a graph such that $S \subseteq V$ and $H \subseteq E$. An *induced* subgraph $G[S], S \subset V$ of a graph $G = (V, E)$ is the graph such that its edge set, denoted by $E[S]$, corresponds to $\bigcup_{uv \in E, u \in S, v \in S} \{u, v\}$.

A *path* is a sequence of successive edges $(u_1u_2, u_2u_3, \ldots, u_{l-1}u_l)$. We denote the path from u_1 to u_l by $u_1 - u_l -$ *path*. A *cycle* is a non-empty path such that $u_1 = u_l$. A graph G is said to be *connected* if there exists a path between every couple of vertices of G. Otherwise, G contains several *connected components*. A *forest* is a graph that contains no cycle. A *spanning tree* is a connected graph with no cycle.

A *bridge* is an edge whose deletion increases the number of connected components by one. An *articulation vertex* is a vertex whose deletion increases the number of connected components. Several uv-paths are *node-disjoint* if every vertices of $V \backslash \{u, v\}$ is visited by at most one path. A graph is said to be *k-node connected* if it contains at least k node-disjoint uv-paths for every pair $(u, v) \in V \times V$. By construction, a graph with no articulation vertex is 2-node connected. Let $W \subset V$, the *cut* induced by W, denoted by $\delta(W)$, corresponds to $uv \in E \mid u \in W, v \in V \backslash W\}$. Given an edge $uv \in E$, *contracting* uv involves deleting the edge uv, merging the two vertices u and v into a unique vertex and keeping the neighbors, creating potentially multiple edges.

Given an oriented graph $G = (V, A)$, a *directed path* is a sequence of successive directed arcs $(u_1u_2, u_2u_3, \ldots, u_{l-1}u_l)$. The *incoming arcs* of a vertex $v \in V$, denoted by $\delta^-(v)$, correspond to the set $\{uv \in A \mid u \in V\}$. The *outgoing arcs* of a vertex $v \in V$, denoted by $\delta^+(v)$,correspond to the set $\{vu \in A \mid u \in V\}$.

For more details on graph theory, the reader can see [12].

2.8 Matroids

Matroids are structures that abstract fundamental properties of dependence commons to graphs and vector spaces. The theory of matroids has many applications in combinatorial optimization. Let S be a finite set and I be a family of subsets of S, $M = (S, I)$ is a *matroid* if the following axioms are satisfied:

1. $\emptyset \in I$
2. If $J' \subseteq J$ and $J \in I$, then $J' \in I$
3. Let $J, J' \in I$ and J' has more elements than J, then there exists $x \in J' \backslash J$ such that $J \cup \{x\} \in I$

The first two properties define an *independent system*. Moreover, subsets of maximum cardinality of I are called *basis*.

The following matroids are well known in the literature. Let S be a finite set and $k \in \mathbb{N}$, $M = (S, I)$, where $J \in I$ if and only if $J \subseteq S$ and $|J| \leq k$, is a *uniform matroid* of rank k. Let $M_i = (S_i, I_i),\ i = 1, \ldots, p$ be uniform matroids of rank k_i on disjoint sets $S_i,\ i = 1, \ldots, p$. $M = (\bigcup_{i=1}^{p} S_i, I)$, where $J \in I$ if and only if $J \subseteq \bigcup_{i=1}^{p} S_i$ and $|J \cap S_i| \leq k_i, i = 1, \ldots, p$, is a *partition matroid* (A partition of a set E is a familly of subsets $E_1, \ldots E_m$ of E such that $E_i \cap E_j = \emptyset, \forall i, j = 1, \ldots, m, i \neq j$ and $\bigcup_{i=1}^{m} E_i = E$). Given a graph $G = (V, E)$, $M = (E, F)$, where $H \in F$ if and only if $H \subseteq E$ and H is the edge set of a forest, is the *graphic matroid* of G.

The *matroid polytope* of a matroid $M = (S, I)$ is the convex hull of the set $\{x^i \mid i$ is a basis of $M\}$. Moreover, the *independence matroid polytope* $P(M)$ is the convex hull of the set $\{x^i |\ i \in I\}$. Given two matroids $M_1 = (S, I_1)$ and $M_2 = (S, I_2)$, Edmonds [24] showed that the *matroid intersection polytope* $P(M_1 \cap M_2) = P(M_1) \cap P(M_2)$ is the convex hull of the set $\{x^i |\ i \in I_1 \cap I_2\}$.

For more details on matroid theory, the reader can see [58].

Chapter 3

The minimum d-budgeted spanning tree problem and polyhedra

In this chapter we give several formulations for the MdBSTP. We investigate the dBST polytope and present conditions under which some inequalities are facet-defining.

3.1 Formulations

Let us denote by P_{ST} the Spanning Tree polytope. An integer programming formulation for the d-budgeted version is given by:

$$\min \sum_{e \in E} c_e x_e \tag{3.1}$$

$$s.t.\ x \in P_{ST}, \tag{3.2}$$

$$\sum_{e \in E} w_e^i x_e \leq B^i, \qquad \forall i = 1, \ldots, d, \tag{3.3}$$

$$x_e \in \{0, 1\}, \qquad \forall e \in E. \tag{3.4}$$

The constraints (3.3) represent the budget constraints, with a binary knapsack constraint structure. The constraints (3.4) are the integrality constraints. Note that there are as many formulations (3.2) − (3.4) as there exist formulations for the STP. In the following, we study some of them in order to find a tight formulation for the d-budgeted case.

Several LP and ILP formulations for the MSTP are introduced in the literature. They all have their own benefits and drawbacks, some of them are integer like the subtour, multicut, multi-flow and Martin's formulations [49], and others are compact but not integer like the flow and Miller-Tucker-Zemlin [53] formulations. We selected among those formulations the ones that seem to be more relevant in both theory and practice, this choice appears to be correlated to their number of variables, constraints and their theoretical properties. For more details on those formulations, the reader can see [48].

3.1.1 Subtour elimination formulation

One of the most famous formulations for the MSTP is based on the so-called *subtour elimination inequalities*. A *subtour* corresponds to cycles in induced subgraphs. The formulation given below, will be denoted by P_{sub}:

$$\min \sum_{e \in E} c_e x_e \tag{3.5}$$

$$s.t. \sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subset V, 2 < |S| < |V| - 1, \tag{3.6}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{3.7}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{3.8}$$

Constraints (3.6) are the *subtour elimination inequalities*, they ensure that there is no cycles in any solution. There is an exponential number of these inequalities, but the related separation problem can be solved in polynomial time. Constraint (3.7) expresses the fact that every spanning tree has exactly $|V| - 1$ edges. Inequalities (3.8) are the trivial constraints. The following result allows one to relax the integrality constraints of the design variables. Let us denote by $\mathcal{H}_{ST}$ the set of incidence vectors of the spanning trees of G.

Proposition 1. *[25] $conv(\mathcal{H}_{ST})$ is given by inequalities (3.6)-(3.8).*

Proposition 1 illustrates the strength of formulation P_{sub}. As the separation problem for inequalities (3.6) can be solved in polynomial time, the MSTP can also be solved in polynomial time by using a cutting plane procedure.

3.1.2 Cut formulation

Another famous formulation of the MSTP, given below, is based on the connectivity constraints, exploiting the tree structure as a connected subgraph. This formulation will be denoted by P_{cut}:

$$\min \sum_{e \in E} c_e x_e \tag{3.9}$$

$$s.t. \sum_{e \in \delta(S)} x_e \geq 1, \qquad \forall S \subset V, 1 \leq |S| \leq |V| - 2, \tag{3.10}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{3.11}$$

$$x_e \in \{0, 1\}, \qquad \forall e \in E. \tag{3.12}$$

The family of constraints (3.10) ensure that for every cut of G, at least one edge in this cut belongs to the solution. These inequalities are called *cut inequalities*. There exists an exponential number of cut inequalities but they can be separated in polynomial time. The

related separation problem reduces to the minimum cut problem. Constraints (3.12) are the integrality constraints. In contrast of the P_{sub} formulation, solving the linear relaxation of P_{cut} can lead to fractional solutions. A branching procedure is required in order to solve the problem. Moreover we have the following property.

Proposition 2. *Let S_{sub} and S_{cut} be respectively the set of solutions related to P_{sub} and P_{cut}. Then $S_{sub} \subseteq S_{cut}$ and in general the inclusion is strict.*

The cut formulation is theoretically less stronger than the subtour formulation but the separation problem of inequalities (3.10) is faster to solve than the one of (3.6). Note that in contrast of the famous *Travelling Salesman Problem* [7], the cut inequalities (3.10) are not equivalent to the subtour ones (3.6). This difference is justified by the 1-connectivity of the spanning tree.

A generalization of the cut formulation, presented above, is the multicut formulation obtained by replacing constraints (3.10) by

$$\sum_{\{u,v\}\in E, u\in V_i, v\in V_j, i\neq j} x_e \geq k-1 \text{ for all partitions } V_1,\ldots,V_k \text{ of } V. \tag{3.13}$$

By construction, we have $S_{mcut} \subseteq S_{cut}$, where S_{mcut} is the set of feasible solutions of the multicut formulation, moreover $S_{mcut} = S_{sub} = conv(\mathcal{H}_{ST})$. The number of partition constraints is huge, even compared to the number of subtour constraints, but the separation problem for these inequalities can be solved in polynomial time [21, 9].

3.1.3 Miller-Tucker-Zemlin formulation

The Miller-Tucker-Zemlin (MTZ) formulation [53] is a compact extended formulation for the MSTP. Additional variables allow one to characterize the acyclic property with a polynomial number of constraints. In this formulation, we consider the directed graph $G_d = (V, A)$ obtained from $G = (V, E)$ by considering two arcs ij and ji for every edge $ij \in E$. The created arcs share the same cost as the original edge in E. Also with every vertex v of G, it is associated a variable u_v, representing the number of edges from v to a root vertex $r \in V$, arbitrarily chosen, in the unique $r - v-$path of the spanning tree solution. The MTZ formulation is given by:

$$\min \sum_{ij\in A} c_{i,j}x_{ij} \tag{3.14}$$

$$s.t. \sum_{i\in\delta^-(j)} x_{ij} = 1, \qquad \forall j \in V\backslash\{r\}, \tag{3.15}$$

$$u_i - u_j + |V|x_{ij} \leq |V| - 1, \qquad \forall ij \in A, j \neq r, \tag{3.16}$$

$$u_i \leq |V| - 1, \qquad \forall i \in V\backslash\{r\}, \tag{3.17}$$

$$u_i \geq 1, \qquad \forall i \in V\backslash\{r\}, \tag{3.18}$$

$$u_r = 0, \tag{3.19}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in V, j \in V\backslash\{i, r\}. \tag{3.20}$$

Constraints (3.15) ensure that every vertex (except the root r) is visited. Inequalities (3.15) - (3.19) ensure that the feasible solutions do not contain cycles. Due to the use of several big-M in constraints (3.16), this formulation has a huge gap between the related relaxation value and the optimal one. But since this formulation is compact, it may be of interest in practice for solving small-sized instances.

3.2 Polyhedral analysis

Let $\mathcal{H}$ be the set of incidence vectors of the feasible solutions of the MdBSTP. In this section, we give several properties of the *d-budgeted spanning tree polytope* $\mathcal{P} = conv(\mathcal{H})$. We start by discussing its dimension and then some facet-defining inequalities.

3.2.1 Dimension

Let $E_x^0 = \{e \in E \mid x_e = 0, \forall x \in \mathcal{H}\}$ and $E_x^1 = \{e \in E \mid x_e = 1, \forall x \in \mathcal{H}\}$ denote respectively the *forbidden* and *necessary* edge sets. Every forbidden (respectively necessary) edge e induces an equation $x_e = 0$ (respectively $x_e = 1$) in the description of $\mathcal{P}$. Hence, the *dimension* of $\mathcal{P}$ is bounded by $|E| - |E_x^0| - |E_x^1|$.

The topology of the graph may also be exploited to obtain tighter bounds. Consider the following example:

Example 1

Consider the graph $G = (V, E)$ given by Figure 3.1.

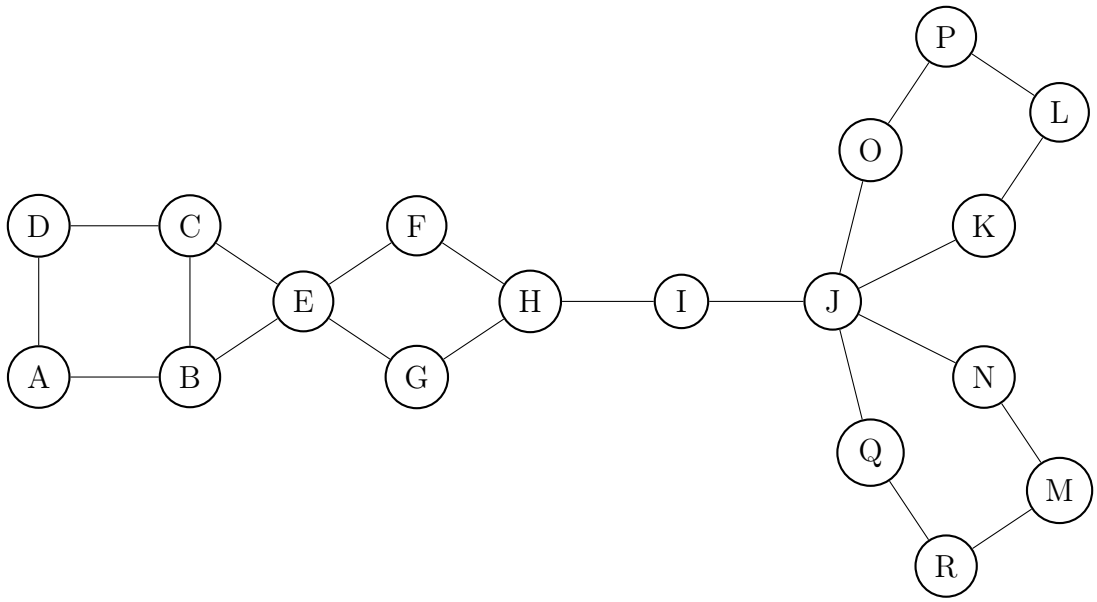


Figure 3.1: A graph with several articulation vertices.

The graph G has 4 articulation vertices: E, H, I and J. We can decomposed the graph into 6 induced subgraphs $G[V_j]$, $j = 1, \ldots, 6$ where $V_1 = \{A, B, C, D, E\}$, $V_2 =$

$\{E, F, G, H\}$, $V_3 = \{H, I\}$, $V_4 = \{I, J\}$, $V_5 = \{J, O, P, L, K\}$ and $V_6 = \{J, Q, R, M, N\}$. Figure 3.2 illustrates those induced subgraphs.

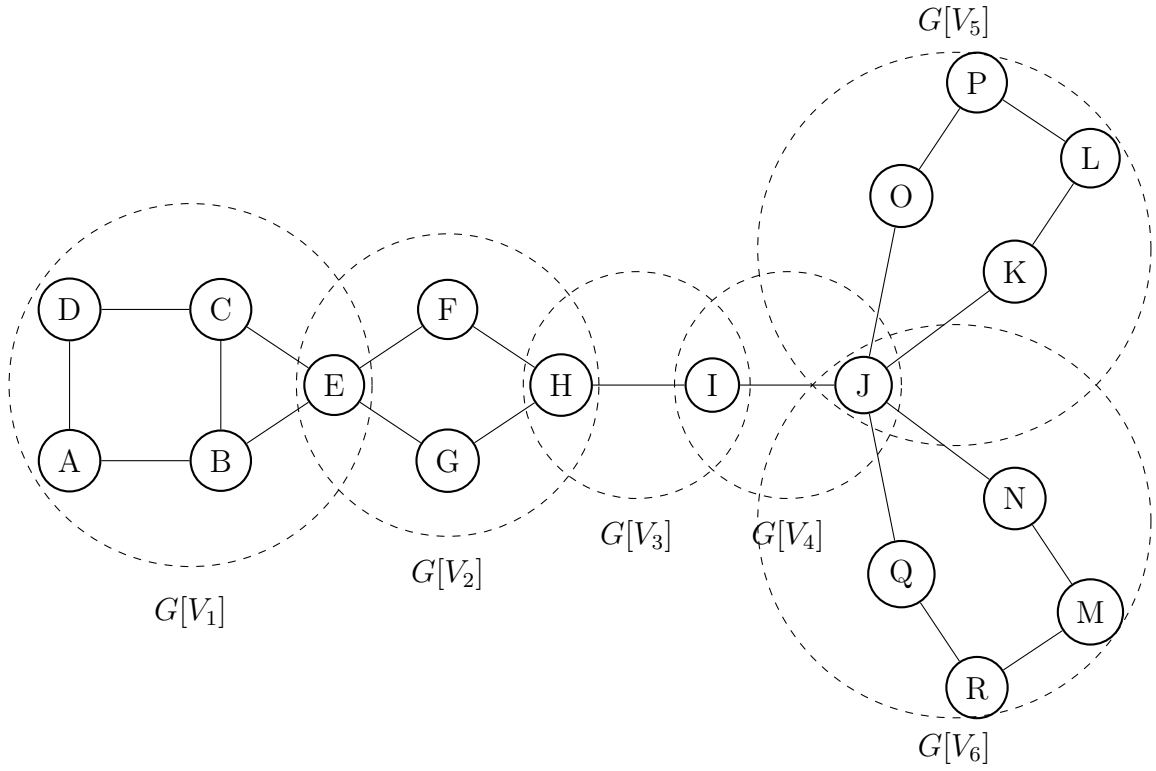


Figure 3.2: Graph decomposition by articulation vertices. Every induced subgraph is circled.

By construction, every spanning tree $T = (V, F)$ of G covers $G[V_j],\ j = 1, \ldots, 6$ with $E[V_j] \cap F = |V_j| - 1$ edges. Indeed, 4 edges are required to cover $G[V_1], G[V_5]$ and $G[V_6]$, 3 edges to cover $G[V_2]$ and exactly one for $G[V_3]$ and $G[V_4]$.

In the general case, given any graph $G = (V, E)$, G can be decomposed into several connected induced subgraphs $G[V_j], j = 1, \ldots, q$ by articulation vertices (also called one-node-cutsets). The resulting induced subgraphs are either 2-node connected graphs or bridges. By construction, every induced subgraph $G[V_j],\ j = 1, \ldots, q$ induces the equation $\sum_{e \in E[V_j]} x_e = |V_j| - 1,\ j = 1, \ldots, q$ in the description of $\mathcal{P}$. All these equalities are linearly independent, and the cardinality equality $\sum_{e \in E} x_e = |V| - 1$ becomes redundant. The dimension of $\mathcal{P}$ is then also bounded by $|E| - q$. The following Proposition exploits both the topology of the graph and the forbidden and necessary edge sets to obtain a tighter bound.

Proposition 3. *$d(\mathcal{P}) \leq |E| - q' - |E_x^0| - |E_x^1|$, where $q' = |\{j \in \{1, \ldots, q\}$ such that $E[V_j] \setminus (E_x^0 \cup E_x^1) \neq \emptyset\}|$.*

Proof. As pointed out, we have $d(\mathcal{P}) \leq |E| - q$ and $d(\mathcal{P}) \leq |E| - |E_x^0| - |E_x^1|$. Moreover, some induced subgraphs $G[V_j],\ j = 1, \ldots, q$ may contain only edges in $E_x^0 \cup E_x^1$, such as

bridges or 2-node connected subgraphs containing several edges with high weight values. In this case the related equalities $\sum_{e\in E[V_j]} x_e = |V_j| - 1$ are redundant with equalities related to E_x^0 and E_x^1. q' corresponds to the number of induced subgraphs such that the equality $\sum_{e\in E[V_j]} x_e = |V_j| - 1$ is not redundant with respect to equations related to E_x^0 and E_x^1. □

In the following, we give a sufficient condition on the budget values of an instance in order to guarantee $d(\mathcal{P}) = |E| - q$. Notice that $\mathcal{P}$ cannot be full-dimensional as there is at least the cardinality equation $\sum_{e\in E} x_e = |V| - 1$. Usually, dimension's proofs consist of enumerating a maximum number of affinely independent incidence vectors. This enumeration may be difficult to obtain given the graph structure and weights. Our approach is based on showing that there are no further equalities than the ones related to the pieces of the graph.

Let $T^w = (V, F^w)$ be a feasible spanning tree for the dBSTP, and let $w^{*i} = \max(\sum_{e\in F^w} w_e^i, w'^i)$ where $w'^i = \max\{\sum_{e\in F^w} w_e^i + w_h^i - w_f^i \mid h \in E\backslash F^w, f \in F^w, (V, F^w \cup \{h\}\backslash\{f\}) \text{ is a spanning tree}\}$, $i = 1, \ldots, d$. The thresholds w^{*i} are computed in such a way that every swap of an edge $f \in F^w$ with an edge $h \in E\backslash F^w$ leads to a feasible solution.

Proposition 4. *If $B^i \geq w^{*i}$, $\forall i = 1, \ldots, d$, then $d(\mathcal{P}) = |E| - q$.*

Proof. Since each induced subgraph $G[V_j]$, $j = 1, \ldots, q$ induces an equality $\sum_{e\in E[V_j]} x_e = |V_j| - 1$ in the description of $\mathcal{P}$, $d(\mathcal{P}) \leq |E| - q$. Now assume that there is a further equation $\sum_{e\in E} a_e x_e = b$ in the description of $\mathcal{P}$, with $b, a_e \in \mathbb{R}, \forall e \in E$, non-redundant with respect to the cardinality equations. We have the following Lemma:

Lemma 1. *Every subgraph $G[V_j]$, $j = 1, \ldots, q$ verifies $a_e = a_f, \forall e, f \in E[V_j]$.*

Proof. Given any subgraph $G[V_j]$, $j = 1, \ldots, q$, assume by contradiction that there are two edges e_1 and $e_2 \in E[V_j]$ such that $a_{e_1} \neq a_{e_2}$. Consider the unique path $u_1, u_2, u_3, u_4, \ldots, u_l$ in $F^w \cap E[V_j]$ such that $u_1u_2 = e_1$ if $e_1 \in F^w$, otherwise if $e_1 \notin F^w$ let u_1u_2 be an edge of $C\backslash\{e_1\}$, where C is the unique cycle created by adding e_1 to T^w. Similarly, let $u_{l-1}u_l = e_2$ if $e_2 \in F^w$, otherwise if $e_2 \notin F^w$, let $u_{l-1}u_l$ be an edge in $C'\backslash\{e_2\}$, where C' is the unique cycle created by adding e_2 to T^w. An example of such a $u_1 - u_l$−path is given in Figure 3.3.

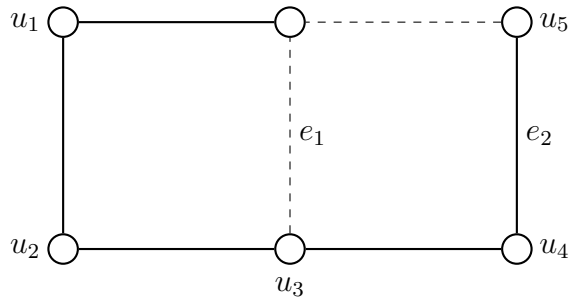


Figure 3.3: Construction of a $u_1 - u_l$−path in $G[V_j]$. Edges in F^w are represented in black and edges in $E[V_j]\backslash F^w$ are dashed.

We will prove by induction that the $r,\ r = 1, \ldots, l-1$, first edges of the $u_1 - u_l-$path have the same coefficient a. The first edge of the $u_1 - u_l-$path is u_1u_2 and it has a unique coefficient $a_{u_1u_2}$. Now assume that the r first edges of the $u_1 - u_l-$path have the same coefficient and consider the r-th and $r+1$-th edges of the path; u_ru_{r+1} and $u_{r+1}u_{r+2}$. As $G[V_j]$ is 2-node connected, by removing vertex u_{r+1} there is still at least one path p in the induced subgraph $G[V_j \backslash \{u_{r+1}\}]$ from u_r to u_{r+2}. Removing u_{r+1} from T^w creates several connected components. By construction u_r and u_{r+2} are in different components but the path p contains edges linking those connected components.

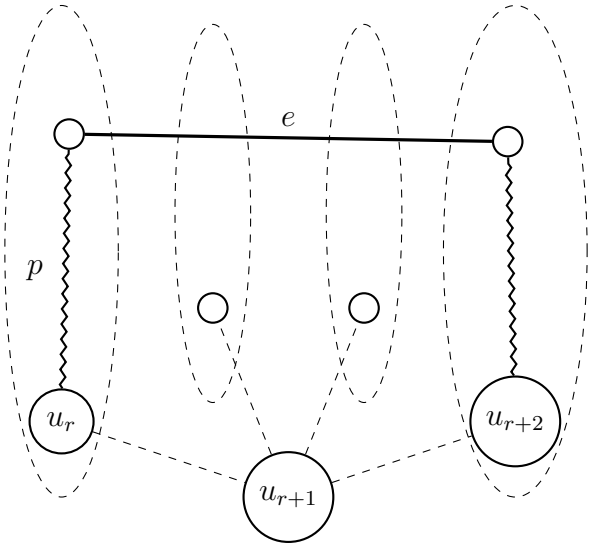


Figure 3.4: Connected components created by removing u_{r+1} from T^w. Each component is represented with a dashed ellipse. The path p is represented in bold. Dashed edges are incident edges of u_{r+1}. Zigzags represent subpaths.

If there exists an edge $e \in E[V_j]$ between the connected component containing u_r and the connected component containing u_{r+2}, as represented in Figure 3.4, by adding e to T^w we create a cycle C formed by the edges in p, u_ru_{r+1} and $u_{r+1}u_{r+2}$. By hypothesis, by swapping any edge of $C \backslash \{e\}$ with e, we obtain a feasible dBST solution. Hence $a_f = a_g, \forall f, g \in C$ and u_ru_{r+1} and $u_{r+1}u_{r+2}$ have the same coefficient.

If not, the path p comes through at least one connected component that does not contain u_r nor u_{r+2}, see Figure 3.5. Let $CC_1, CC_2, \ldots, CC_t$ be the ordered visited connected components by p, where CC_1 is the connected component containing u_r and CC_t is the connected component containing u_{r+2}. Notice that there exists a path p that goes through each component at most once, otherwise there should exist a cycle formed by p and edges of a given component $CC_{t'},\ t' \in \{1, \ldots, t\}$ and here we can modify p in such a way that it goes at most once through each component.

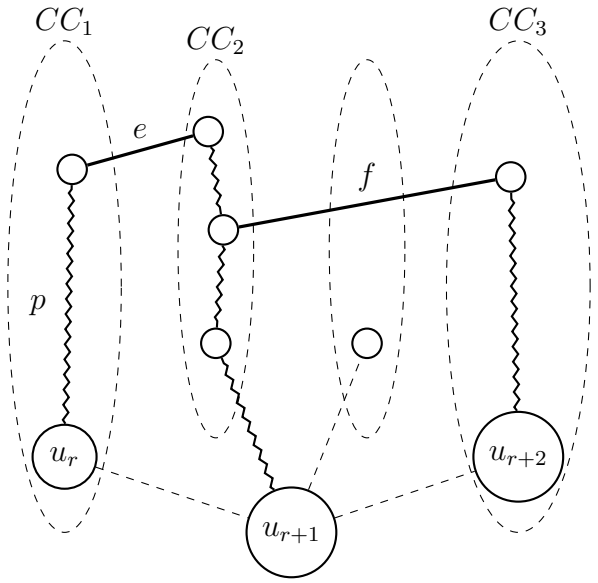


Figure 3.5: Connected components created by removing u_{r+1}. Each component is represented with a dashed ellipse. The path p is represented in bold. Zigzags represent subpaths.

Consider three consecutive components $CC_{t'}, CC_{t'+1}$ and $CC_{t'+2}$ for $t' = 1, \ldots, t-2$, and let e (respectively f) be the edge in p connecting $CC_{t'}$ and $CC_{t'+1}$ (respectively $CC_{t'+1}$ and $CC_{t'+2}$). By adding e to T^w, we create a cycle C_e that contains e and two edges in $\delta(u_{r+1})$. By swapping any edge of $C_e \backslash \{e\}$ with e, we obtain a feasible solution. Hence $a_h = a_g, \forall h, g \in C_e$. Moreover, by adding f to T^w, we create a cycle C_f that contains f and two edges in $\delta(u_{r+1})$. By swapping any edge of $C_f \backslash \{f\}$ with f, we obtain a feasible solution. Hence $a_h = a_g, \forall h, g \in C_f$. Furthermore, as both cycles share a common edge in $\delta(u_{r+1})$, linking u_{r+1} to $CC_{t'+1}$, we deduce that $a_h = a_g, \forall h, g \in C_e \cup C_f$. By iterating on triplets of consecutive components, we obtain $a_{u_r u_{r+1}} = a_{u_{r+1} u_{r+2}}$.

In consequence, we obtain that every edge of the $u_1 - u_l-$path has the same coefficient. Moreover $a_{u_1 u_2} = a_{e_1}$ (respectively $a_{u_{l-1} u_l} = a_{e_2}$) as either $u_1 u_2 = e_1$ (respectively $u_{l-1} u_l = e_2$) or swap $u_1 u_2$ with e_1 (respectively swap $u_{l-1} u_l$ with e_2) leads to a feasible solution. Hence, $a_{e_1} = a_{e_2}$, contradicting our Lemma's assumption.

□

Considering the result of Lemma 1, every equation in the description of $\mathcal{P}$ has the following structure $\sum_{j=1}^{q} a_j x(E[V_j]) = b$ with $b, a_j \in \mathbb{R}, \forall j = 1, \ldots, q$. Which is redundant with respect to the cardinality equalities related to the subgraphs $G[V_j], j = 1, \ldots, q$. □

The condition on T^w for Proposition 4 is not restrictive in practice, especially if the instance has a reasonable size. For the special case where $d = 1$, the best choice for T^w is a spanning tree that minimizes $\sum_{e \in F^w} w_e^1$. Notice that the condition given in Proposition 4 is only a sufficient condition. The converse might not be true. Figure 3.6 shows a counterexample with weights on the edges and a budget of 28:

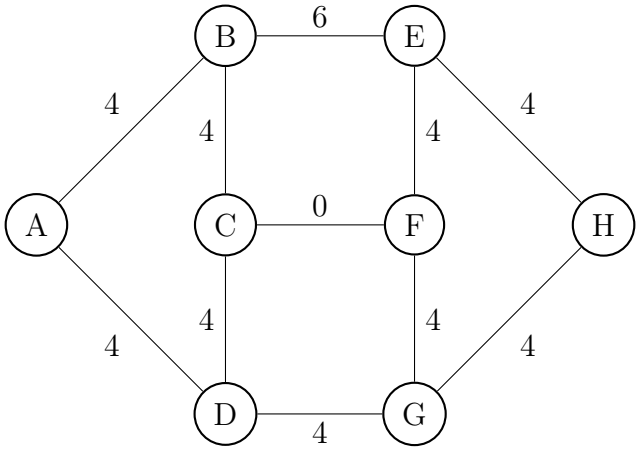


Figure 3.6: Counterexample.

For the instance given by Figure 3.6, $d(\mathcal{P}) = |E[-1$ with a budget of 28, although our threshold is $w^{*1} = 30$.

3.2.2 Facial aspect

The study of the dimension is a crucial point in order to obtain conditions on facet-defining inequalities. Facet-defining inequalities belong to the tightest descriptions of $\mathcal{P}$. Incorporate those inequalities in a branching algorithm may improve its running time. We prove that several inequalities of the subtour formulation define facets for the dBST polytope $\mathcal{P}$. First, we consider the trivial inequalities:

Proposition 5. *Let $e \in E$, then the inequality $x_e \geq 0$ is facet-defining if:*

1. *Every component $G[V_j] = (V_j, E[V_j]\backslash\{e\}),\ j = 1, \ldots, q$ is either a bridge or 2-node connected subgraph,*

2. *There exits a feasible solution $T^w = (V, F^w)$ such that $e \notin F^w$ and $B^i \geq \max\{\sum_{f \in F^w} w_f^i,\ \max\{\sum_{f \in F_w} w_f^i + w_h^i - w_g^i \mid h \in E\backslash F^w, g \in F^w, (V, F^w \cup \{h\}\backslash\{g\})$ is a spanning tree$\}\},\ i = 1, \ldots, d$.*

Proof. First, we show that $\{x \in \mathcal{P} \mid x_e = 0\} \neq \emptyset$. As $e \notin F^w$, then the incidence vector of T^w belongs to $\{x \in \mathcal{P} \mid x_e = 0\}$. Next, we show that $\{x \in \mathcal{P} \mid x_e = 0\} \neq \mathcal{P}$. By adding e to T^w, we create a cycle. We can swap e with any edge of the created cycle in order to get a feasible solution not in $\{x \in P \mid x_e = 0\}$. Now, we show that the dimension of $\{x \in \mathcal{P} \mid x_e = 0\}$ is equal to $d(\mathcal{P}) - 1$. By construction of T^w and Proposition 4, the polytope related to the instance verifies $d(\mathcal{P}) = |E| - q$. By applying Lemma 1 to $G' = (V, E\backslash\{e\})$ and T^w, we have $a_g = a_h,\ \forall g, h \in E[V_j]\backslash\{e\}$ for every induced subgraph $G[V_j] = (V_j, E[V_j]\backslash\{e\}),\ j = 1, \ldots, q$. This implies that there is no equation non-redundant with respect to $x_e = 0$ and $\sum_{f \in E[V_j]\backslash\{e\}} x_f = |V_j| - 1,\ \forall j = 1, \ldots, q$ in the description of $\{x \in \mathcal{P} \mid x_e = 0\}$. Then the dimension of $\{x \in \mathcal{P} \mid x_e = 0\}$ is equal to $d(\mathcal{P}) - 1$. □

Proposition 6. *Let $e \in E$, then the inequality $x_e \leq 1$ is facet-defining if:*

1. *Every component* $G[V_j] = (V_j, E[V_j]),\ j = 1, \ldots, q$ *where* $e \notin E[V_j]$ *is either a bridge or a 2-node connected subgraph,*

2. *The component* G/e *obtained by contracting* e *in* $G[V_j] = (V_j, E[V_j])$ *with* $e \in E[V_j]$ *is either a bridge with multi-edges or a 2-node connected subgraph,*

3. *There exists a feasible solution* $T^w = (V, F^w)$ *such that* $e \in F^w$ *and* $B^i \geq \max\{\sum_{f \in F^w} w_f^i, \max\{\sum_{f \in F_w} w_f^i + w_h^i - w_g^i \mid h \in E \backslash F^w, g \in F^w, (V, F^w \cup \{h\} \backslash \{g\})$ is a spanning tree $\}\},\ i = 1, \ldots, d.$

Proof. First, we show that $\{x \in \mathcal{P} \mid x_e = 1\} \neq \emptyset$. As $e \in F^w$, then the incidence vector of T^w belongs to $\{x \in \mathcal{P} \mid x_e = 1\}$. Next, we show that $\{x \in \mathcal{P} \mid x_e = 1\} \neq \mathcal{P}$. By hypothesis, we can swap e with any edge in the empty cut induced by removing e from T^w. Such an edge exists as the component G/e is either a bridge with multi-edges or a 2-node connected subgraph. Now, we show that the dimension of $\{x \in \mathcal{P} \mid x_e = 1\}$ is equal to $d(\mathcal{P}) - 1$. By construction of T^w and Proposition 4, the polytope related to the instance verifies $d(\mathcal{P}) = |E| - q$. Any connected component $G[V_j]$ $j = 1, \ldots, q$ with $e \notin E_j$ verifies $a_g = a_h, \forall g, h \in E[V_j]$ by Lemma 1. G/e may contains multi-edges but several multi-edges cant belong to T^w, otherwise there will be a cycle with e. Hence, the proof of Lemma 1 remains valid for G/e and we have $a_g = a_h, \forall g, h \in E[V_j] \backslash \{e\}$ for the component $G[V_j]$ with $e \in E[V_j]$. This implies that there is no equation non-redundant with respect to $x_e = 1$ and $\sum_{f \in E[V_j]} x_f = |V_j| - 1, \forall j = 1, \ldots, q$ in the description of $\{x \in \mathcal{P} \mid x_e = 1\}$. Therefore the dimension of $\{x \in \mathcal{P} \mid x_e = 1\}$ is equal to $d(\mathcal{P}) - 1$. □

Now we discuss the facial aspect of the subtour inequalities.

Proposition 7. *Let* $S \subset V$*, then the inequality* $x(E[S]) \leq |S| - 1$ *is facet-defining if:*

1. *The subgraph* $G[S]$ *is 2-node connected,*

2. *Every subgraph* $G[V_j] = (V_j, E[V_j]),\ j = 1, \ldots, q$ *where* $S \cap V_j = \emptyset$ *is either a bridge or a 2-node connected graph,*

3. *The subgraph* G/S *obtained by contracting* $E[S]$ *in* $G[V_j] = (V_j, E[V_j])$ *where* $S \subset V_j$ *is either a bridge with multi-edges or a 2-node connected component,*

4. *There exists a feasible solution* $T^w = (V, F^w)$ *such that* $|F^w \cap E[S]| = |S| - 1$ *and* $B^i \geq \max\{\sum_{f \in F^w} w_f^i, \max\{\sum_{f \in F^w} w_f^i + w_h^i - w_g^i \mid h \in E \backslash F^w, g \in F^w, (V, F^w \cup \{h\} \backslash \{g\})$ is a spanning tree$\}\},\ i = 1, \ldots, d.$

Proof. First, we show that $\{x \in \mathcal{P} \mid \sum_{e \in E[S]} x_e = |S| - 1\} \neq \emptyset$. By hypothesis, T^w is a feasible solution, then the incidence vector of T^w belongs to $\{x \in \mathcal{P} \mid \sum_{e \in E[S]} x_e = |S| - 1\}$. Next, we show that $\{x \in \mathcal{P} \mid \sum_{e \in E[S]} x_e = |S| - 1\} \neq \mathcal{P}$. Indeed, by swapping an edge $e \in F^w \cap E[S]$ with an edge $f \notin S$ in the empty cut induced by removing e from T^w, such an edge exists as G/S is either a bridge with multi-edges or a 2-node connected subgraph. Now, we show that the dimension of $\{x \in \mathcal{P} \mid \sum_{e \in E[S]} x_e = |S| - 1\}$ is equal to $d(\mathcal{P}) - 1$. By construction of T^w and Proposition 4, the polytope related to the instance verifies $d(\mathcal{P}) = |E| - q$. Assume that there is an equation non-redundant with respect to $\sum_{e \in E[S]} x_e = |S| - 1$ and $\sum_{e \in E[V_j]} x_e = |V_j| - 1,\ \forall j = 1, \ldots, q$. We show that:

1. $a_f = a_g,\ \forall f, g \in E_j$ for every subgraph $G[V_j] = (V, E[V_j]),\ j = 1, \ldots, q$ where $V_j \cap S = \emptyset$,
2. $a_f = a_g,\ \forall f, g \in E[S]$,
3. $a_f = a_g,\ \forall f, g \in E_j \backslash E[S]$ for the subgraph $G[V_j] = (V, E[V_j])$ where $S \subset V_j$.

The first statement comes from Lemma 1. For the second one, we consider the induced graph $G[S] = (S, E[S])$ and we apply Lemma again 1 on $G[S]$. For the third statement, we consider the graph G/S. G/S may contains multi-edges but several multi-edges cant belong to T^w, otherwise there will be a cycle with $E[S] \cap F^w$. Hence, the proof of Lemma 1 remains valid for G/S. The three statements imply that there is no equation non-redundant with respect to $\sum_{e \in E[S]} x_e = |S| - 1$ and $\sum_{e \in E[V_j]} x_e = |V_j| - 1,\ \forall j = 1, \ldots, q$ in the description of $\{x \in \mathcal{P} \mid \sum_{e \in E[S]} x_e = |S| - 1\}$. Therefore the dimension of $\{x \in P \mid \sum_{e \in E[S]} x_e = |S| - 1\}$ is equal to $d(\mathcal{P}) - 1$. □

In the case $d = 1$, the spanning tree T^w in the hypothesis of Propositions 5, 6 and 7 can be obtained by computing and adapting a minimum weight spanning tree.

Also the subtour inequalities may be facet-defining, the subtour formulation may be efficient for solving the MdBSTP by Branch-and-Cut.

3.3 Special cases of the d-budgeted spanning tree problem

The MdBSTP lies at the intersection of the STP and the KP. By adding more structure to the knapsack part of the problem it may be possible to exploit it in order to solve the MdBSTP in polynomial time. A first special case, is given below:

Proposition 8. *If $d = 1$ and $w_e^1 \in \{0, 1\},\ \forall e \in E$, then the polytope described by the following inequalities (3.21)-(3.24) is integral.*

$$\sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subset V, 2 < |S| < |V| - 1, \tag{3.21}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{3.22}$$

$$\sum_{e \in E} w_e^1 x_e \leq B^1, \tag{3.23}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{3.24}$$

Proof. Assume that x^* is a fractional extreme point of (3.21)-(3.24). The subtour polytope (3.21, 3.22, 3.24) is integer, then x^* must satisfy at equality the budget constraint (3.23). Moreover, by the adjacency of the basis of the extreme points in the spanning tree polytope, x^* is a convex combination of two incidence vectors x^1 and x^2 associated with two spanning trees $T^1 = (V, F^1)$ and $T^2 = (V, F^2)$ that differ by exactly one edge. We deduce that $B^1 = \sum_{e \in F^1} w_e < \sum_{e \in F^2} w_e = B^1 + 1$. Since $B^1 \in \mathbb{N}$, then x^* does not satisfy the constraint (3.23), a contradiction. □

In this special case, the problem can then be solved in polynomial time. Proposition 8 can be generalized to a family of special cases; a knapsack constraint where the coefficients are binary corresponds to a *matroid* $M_1 = (E, I_1)$ such that I_1 is a collection of subsets of E with at most B^1 edges among $\{e \in E \mid w_e = 1\}$. Another well-known matroid is the *graphic matroid* $M_2 = (E, I_2)$, where every element of I_2 is a forest of E. Every basis of the graphic matroid corresponds to a spanning tree of G. The intersection $I_1 \cap I_2$ is the collection of forests that satisfy the budget constraint associated with M_1. Moreover, the *independence polytope* $P(M_1 \cap M_2)$ is integer and corresponds to $P_1 \cap P_2$ where P_1 (respectively P_2) is the independence matroid polytope of M_1 (respectively M_2) [24]. The following Proposition generalizes the previous special case:

Proposition 9. *If the budget inequality defines a matroid, then the following linear program is integer:*

$$\min \sum_{e \in E} c_e x_e \tag{3.25}$$

$$s.t. \sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subset V, 2 < |S| < |V| - 1, \tag{3.26}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{3.27}$$

$$\sum_{e \in E} w_e^1 x_e \leq B^1, \tag{3.28}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{3.29}$$

Proof. The polytope $P(M_G \cap M_B)$ described by inequalities (3.26),(3.28) and (3.29) corresponds to the matroid intersection polytope related to the graphic matroid M_G and the matroid M_B associated with the budget inequality 3.28. Hence, the polyhedron $P(M_G \cap M_B)$ is integer. The polytope P_{sub} described by inequalities (3.26) − (3.29) is a proper face of $P(M_G \cap M_B)$ then it is also integral. □

Every extreme point of the polytope (3.26)-(3.29) corresponds to a spanning tree that satisfies the budget constraint 3.28. The budget structure can be seen as a matroid in some particular cases. Some algorithms have been developed to identify if a specific knapsack constraint defines a matroid [4], [14]. In [14], the authors have devised a polynomial algorithm, based on Wolsey's necessary and sufficient conditions for a knapsack constraint to be a matroid [78].

Chapter 4

Linear and Lagrangian relaxations for the minimum 1-budgeted spanning tree problem

The following chapters will be devoted to the special case of the d-budgeted problem where $d = 1$. Denoting by $\mathcal{P}_i = conv(\{x \in \{0;1\}^{|E|} \mid x \in P_{sub}, \sum_{e \in E} w_e^i x_e \leq B^i\})$ for $i = 1, \ldots, d$ the integer 1BST polytopes related to each resource constraint, we have $\mathcal{P} \subseteq \bigcap_{i=1,\ldots,d} \mathcal{P}_i$. Hence, every valid inequality for a given polytope $\mathcal{P}_i$ is also valid for the d-budgeted polytope $\mathcal{P}$. We hope that strong valid inequalities for every $\mathcal{P}_i$ will be also effective for the general case. Among the most successful methods for solving integer programs, there are the linear programming based Branch-and-Bound algorithms when the underlying linear programs are strengthened by cutting planes. In some cases, deep cuts could be obtained by solving the linear program relaxation of the problem. Therefore, it is crucial to solve efficiently the linear program relaxation. In the previous chapter, several formulations of the problem were discussed. In the sequel, we focus on the subtour elimination linear program as it is one of the tightest formulations. The number of constraints is exponentially large. However, there exists a polynomial time separation oracle requiring $O(|V|)$ maximum flow problems [59]. We provide computational experiments showing that this algorithm is time consuming even for medium sized instances. Goemans and Ravi [65] present a combinatorial algorithm to solve in strongly polynomial time the linear relaxation of the problem. Their algorithm is based on the Lagrangian relaxation technique and Megiddo's parametric search technique [51]. By using an $O(|E|log(|E|))$ algorithm to sort the edge costs, an optimal solution could be computed by solving $O(|E|log(|E|))$ minimum spanning trees problems. A faster algorithm could be obtained by using a more sophisticated sorting algorithms. However, these parallel algorithms require several cores being implemented. We present a practical efficient algorithm based on a simple binary search algorithm combined with a pruning technique that allows one to substantially reduce the size of the problem.

4.1 Linear programming relaxation

The linear programming relaxation of the subtour elimination formulation of the 1BSTP is obtained by dropping all the integrality constraints.

$$\min \sum_{e \in E} c_e x_e \tag{4.1}$$

$$s.t. \sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subset V, 2 < |S| < |V| - 1, \tag{4.2}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{4.3}$$

$$\sum_{e \in E} w_e x_e \leq B, \tag{4.4}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{4.5}$$

Adding the budget constraint (4.4) breaks the combinatorial structure of the spanning tree polytope (4.2), (4.3) and (4.5). A known result in linear programming [11] states that in this case, every extreme point of (4.2)-(4.5) is either an extreme point of the spanning tree polyhedron, i.e., the incidence vector of a spanning tree, or a convex combination of two adjacent extreme points of the spanning tree polytope. The latter case is illustrated in Figure 4.1.

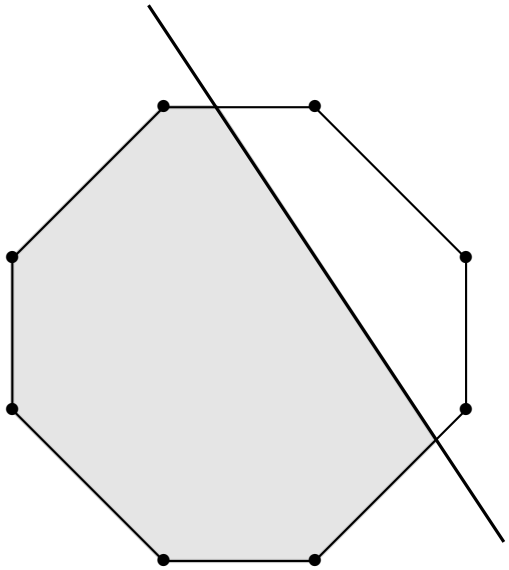


Figure 4.1: Impact of a budget constraint to the spanning tree polytope. The gray area represents the set of solutions satisfying both the spanning tree and the budget constraints.

The concept of adjacent spanning trees is useful in our context. Two spanning trees are said to be *adjacent* if both share all but an edge [76]. As a consequence, every fractional extreme point of the linear relaxation (4.2)-(4.5) has exactly $|V|$ variables with strictly positive values and exactly two fractional variables. Moreover, given a fractional optimal solution x^* of the problem (4.1)-(4.5), the support graph $G^* = (V, E^*)$ where $E^* = \{e \in$

$E \mid x_e^* > 0\}$ is a spanning tree augmented with an edge. In the unique cycle, two edges have fractional values.

Let x^* be an optimal solution of the problem (4.1)-(4.5). If x^* is integer, then it is an optimal solution of the 1BSTP. Otherwise it is a combination of the incidence vector of $T^< = (V, F^<)$ a spanning tree satisfying the knapsack constraint (4.4) and $T^> = (V, F^>)$ a spanning tree violating the knapsack constraint. Moreover, we have the following:

$$\sum_{e \in F^>} c_e < \sum_{e \in E} c_e x_e^* \leq \sum_{e \in E} c_e x'_e \leq \sum_{e \in F^<} c_e, \tag{4.6}$$

where x' is an optimal solution of arg min $\{\sum_{e \in E} c_e x_e \mid x \in \{0,1\}^{|E|},\ x$ satisfies $(4.2) - (4.4)\}$. As both spanning trees have one edge difference, the following is satisfied:

$$\sum_{e \in F^<} c_e - \sum_{e \in F^>} c_e \leq \max\{c_e \mid e \in E\} - \min\{c_e \mid e \in E\}. \tag{4.7}$$

In the literature, it is well known that $T^<$ is a 2-approximate solution. By computing and rounding the linear relaxation optimal solution, one can get a feasible solution with a cost value close to the optimal one. This rounded solution can be used as a MIP Start in order to speed up the problem resolution.

Corollary 1. *In the particular case where the costs $c_e \in \{0,1\},\ \forall e \in E$, the incidence vector of $T^<$ is an optimal solution of arg min $\{\sum_{e \in E} c_e x_e \mid x \in \{0,1\}^{|E|},\ x$* satisfies (4.2)$-(4.4)\}$.

The LP relaxation (4.1)-(4.5) can be solved in polynomial time as the subtour elimination constraints (4.2) can be separated in polynomial time. Even though the linear program can be solved in polynomial time, this separation algorithm can be time consuming. Given a fractional solution x^*, the separation algorithm needs to find a set $S \subset V,\ 2 < |S| < |V| - 1$ such that $\sum_{e \in E[S]} x_e^* > |S| - 1$ if such a set exists. The following algorithm either finds violated subtour constraints associated with connected component or calls the separation algorithm [59].

Algorithm 2: Separation of subtour inequalities

Data: $G = (V, E)$, $x^* \in [0:1]^{|E|}$

Consider the support graph $G^* = (V, E^*)$ where $E^* = \{e \in E \mid x^*(e) > 0\}$;
if x^* is integer **then**
  if there are $p > 1$ connected components in G^* **then**
    for every connected component $G^*[V_i]$, $i = 1, \ldots, p$ **do**
      if component $G^*[V_i]$ contains more than $|V_i| - 1$ edges **then**
        Add the subtour constraint $\sum_{e \in E[V_i]} x_e \leq |V_i| - 1$;
      end
    end
  end
else
  if there are $p > 1$ connected components in G^* **then**
    for every connected component $G^*[V_i]$, $i = 1, \ldots, p$ **do**
      if $\sum_{e \in E[V_i]} x^*_e > |V_i| - 1$ **then**
        Add the subtour constraint $\sum_{e \in E[V_i]} x_e \leq |V_i| - 1$;
      end
    end
  else
    Run Algorithm 3(G^*);
    Run separation algorithm from [59];
  end
end

Algorithm 2 considers several cases. First, if $x^* \in \{0,1\}^{|E|}$, then by the cardinality constraint (4.3), G^* contains exactly $|V| - 1$ edges. If it contains only one connected component, then x^* is a spanning tree (no violated subtour inequality). Otherwise (G^* has two or more connected components), using Breadth-First Search, the algorithm detects all the connected components and adds a violated subtour inequality associated with each connected component having a cycle. In contrast, if x^* is fractional and G^* has two or more connected components, the algorithm checks all the connected components in G^* and outputs the violated associated subtour constraints. If G^* contains exactly one connected component, the algorithm calls the separation procedure of Padberg and Wolsey [59]. Note that the bridges of G^* cannot belong to any cycle. Therefore, one could apply Algorithm 3 in order to reduce the size of the support graph as well as the running time of the algorithm [59]. Algorithm 3 removes recursively edges of degree 1 in the graph, we will denote the resulting deleted edge set by *branches*. Algorithm 3 is very fast in practice and requires $O(|V|^2)$ times. This step is particularly helpful in the case where the separation algorithm [59] is executed as its complexity is $O(|V|^4)$.

Algorithm 3: Deletion of all branches of a graph

```
Data: G* = (V, E*)
for every vertex v in V do
    while v has exactly one neighbor u in G* do
        Delete the edge {u, v} from E*;
        v ← u
    end
end
```

Figure 4.2 gives an example of a connected support graph. Applying Algorithm 3 to the resulting support graph removes the branches $\{AB, BC, BD\ , BE, FH, LN, LO\}$ from E^*.

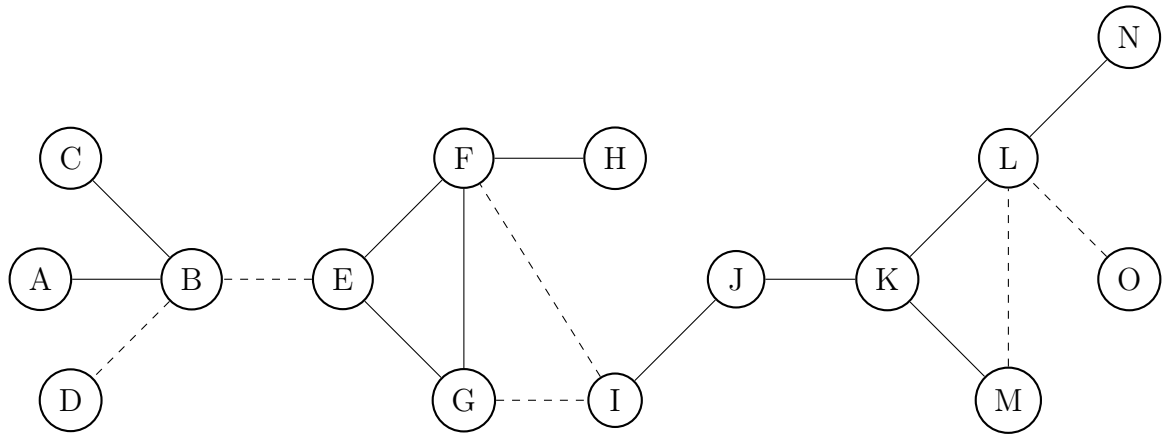


Figure 4.2: Example of a connected support graph associated with a infeasible fractional solution x^*. A dashed edge e corresponds to a fractional variable with value $x^*_e = \frac{1}{2}$ and a bold edge e corresponds to an integer variable with value $x^*_e = 1$. Edges IJ and JK are bridges but they do not belong to a branch as they are between two 2-node induced subgraphs.

Nevertheless, despite the improvement provided by those heuristics, solving the LP relaxation (4.1)-(4.5) may be still time consuming as several calls of the separation algorithm may be necessary. The next section discusses a faster algorithm based on Lagrangian relaxation.

4.2 Lagrangian relaxation

The Lagrangian relaxation is a standard technique well suited for problems where the set of constraints can be divided into 2 sets:

- "good" constraints for which the problem is easily solvable (like the subtour formulation of the STP),
- "bad" constraints that make the problem hard (like the budget constraint).

The main idea is to relax the problem by removing the "bad" constraints and putting them in the objective function with an assigned weight. The latter represents a penalty of not satisfying the constraints. Consider the following problem:

$$\max_{z\geq 0} \; L(z) = \min \sum_{e\in E} c_e x_e + z(\sum_{e\in E} w_e x_e - B) \tag{4.8}$$

$$\sum_{e\in E[S]} x_e \leq |S| - 1, \qquad \forall S \subset V, 2 < |S| < |V| - 1, \tag{4.9}$$

$$(LR) \qquad \sum_{e\in E} x_e = |V| - 1, \tag{4.10}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{4.11}$$

Since the spanning tree polytope (4.9)-(4.11) is integral, the optimal objective function value of the Lagrangian problem (4.8)-(4.11) equals the optimal objective value function of the linear program (4.1)-(4.5) [79]. The composite cost $\sum_{e\in F} c_e + z(\sum_{e\in F} w_e - B)$ for any spanning tree $T = (V, F)$ is a linear function of z with an intercept $\sum_{e\in F} c_e$ and a slope $(\sum_{e\in F} w_e - B)$. In Figure 4.3, we have plotted each of these spanning tree composite costs. Note that for any specific value of the Lagrangian multiplier z, $L(z)$ can be computed by evaluating each composite cost function (line) and identifying the one with the minimum cost. Therefore, the Lagrangian multiplier function $L(z)$ is the lower envelope of the composite lines and the highest point of this envelope at z^* corresponds to the optimal solution of the Lagrangian problem.

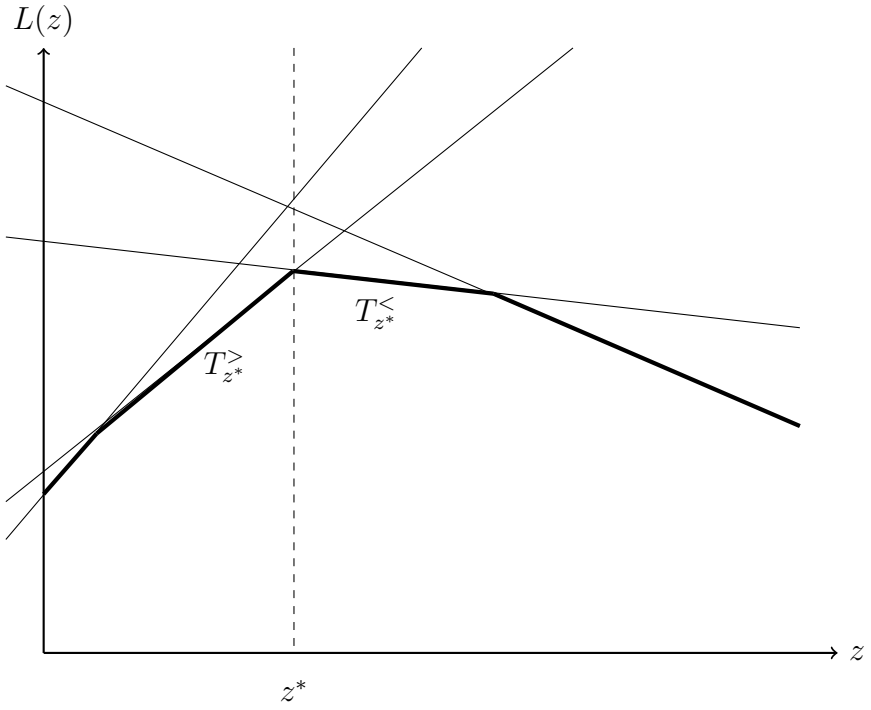


Figure 4.3: Composite cost functions of spanning trees and Lagrangian relaxation. The lower envelope of the composite lines, corresponding to optimal solutions of $L(z)$, $z \geq 0$, is represented in bold.

By construction, a spanning tree satisfying the budget constraint is associated with a non-increasing linear composite cost function and a spanning tree violating the budget constraint is associated with a non-decreasing linear composite cost function. Due to the concavity of $L(z)$, any spanning tree T_z, optimal solution for $L(z)$ with $z \in [0; z^*[$, does not satisfy the budget constraint. Moreover, any optimal solution T_z for $L(z)$ with $z \in]z^*; +\infty[$ satisfies the budget constraint. The two spanning trees $T_{z^*}^{>}$ and $T_{z^*}^{<}$ such that their related composite cost functions intersect at z^*, are optimal solutions of the Lagrangian relaxation at z^*.

4.2.1 Solving efficiently the Lagrangian relaxation

In order to solve efficiently the Lagrangian relaxation, we use, as in [65], the parametric search algorithm of Megiddo [51]. The key point of this approach is to compute an optimal spanning tree for the edge composite cost $c_e + z^* w_e$ at the unknown optimal value z^*. As the composite cost of any spanning tree corresponds to the sum of all its edge's composite cost functions. To this end, it is necessary to be able to order these edge composite costs. However, the difficulty is that the edge composite costs $c_e + zw_e$, when z varies in $\mathbb{R}_+$, are partially ordered. See Figure 4.4.

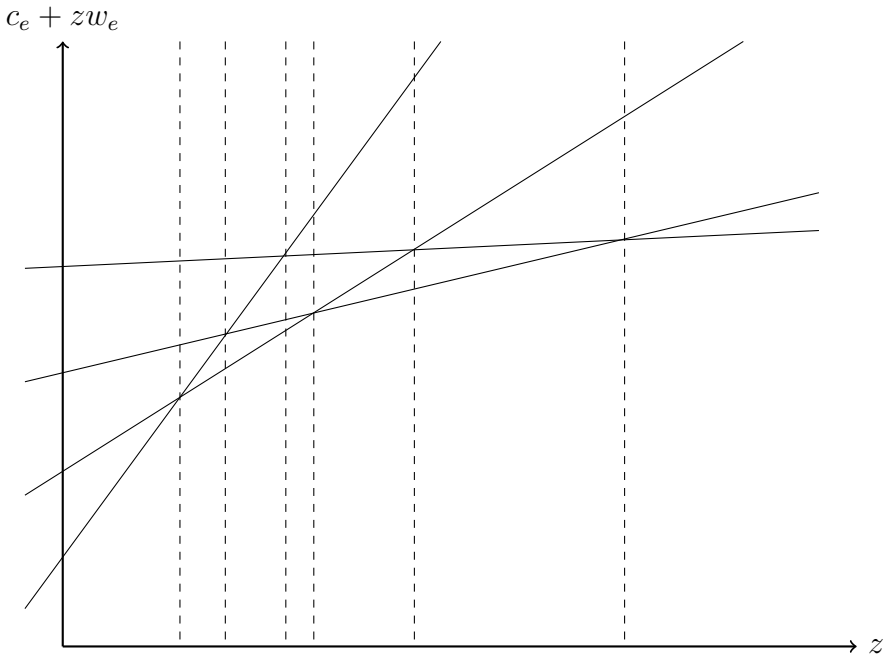


Figure 4.4: Edge composite cost functions. Dashed lines represent z-coordinates corresponding to the intersections of edge composite cost functions. Each created interval represents a unique permutation of E, sorted in a non-decreasing order of $c_e + zw_e$ value.

Given two edges e and f, Megiddo's algorithm can determine if $c_e + z^* w_e = c_f + z^* w_f$, $c_e + z^* w_e > c_f + z^* w_f$ or $c_e + z^* w_e < c_f + z^* w_f$ for the unknown z^* value. For this purpose, the algorithm computes two particular minimum spanning trees for the edge composite costs $c_h + z_{ef} w_h$, where z_{ef} denotes the intersection point of the lines $c_e + z w_e$ and $c_f + z w_f$ as a function of z. More precisely, among all the minimum spanning trees, the algorithm picks the one with the minimum slope $T^<_{z_{ef}} = (V, F^<_{z_{ef}})$ and the maximum slope $T^>_{z_{ef}} = (V, F^>_{z_{ef}})$. See Figure 4.5. This can be done, for instance, by computing two minimum spanning trees for edge composite costs $c_h + (z_{ef} + \epsilon) w_h$ and $c_h + (z_{ef} - \epsilon) w_h$, where $0 < \epsilon << 1$.

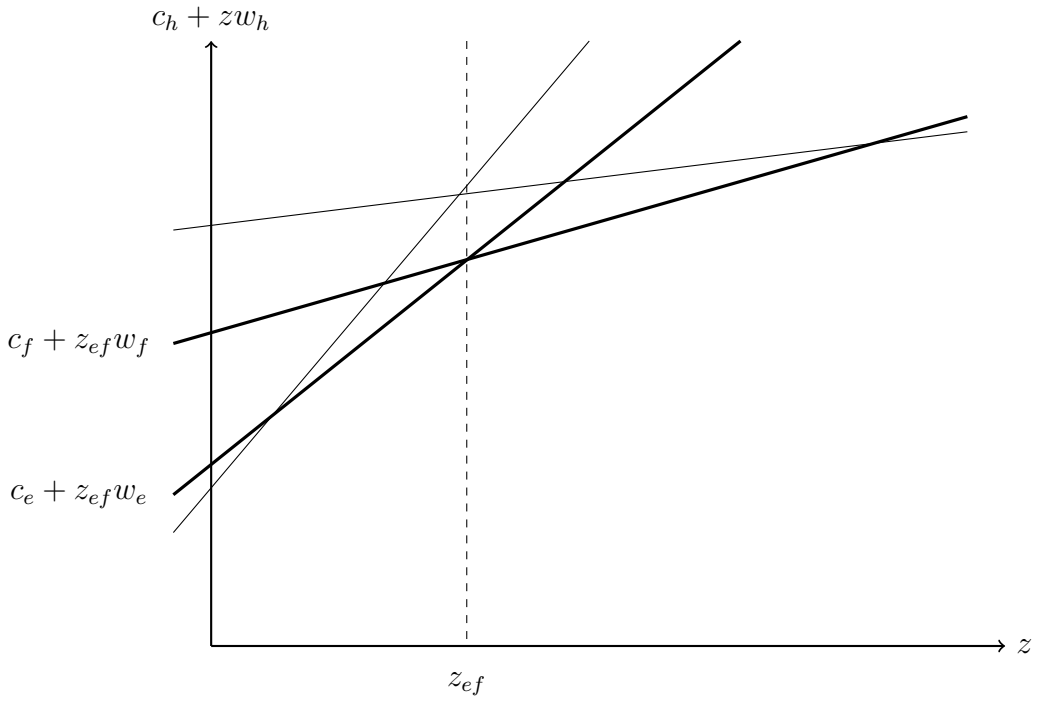


Figure 4.5: Comparing e and f at z_{ef}. The edge composite cost functions related to e and f are represented in bold.

If $\sum_{h \in F^<_{z_{ef}}} w_h > B$ (respectively $\sum_{h \in F^>_{z_{ef}}} w_h < B$) then $z_{ef} < z^*$ (respectively $z_{ef} > z^*$) and $z_{ef} = z^*$ otherwise. By using an algorithm which makes $O(|E| log(|E|))$ comparisons to sort the edge composite costs $c_h + z w_h$, the optimal value z^* can be obtained by solving $O(|E| log(|E|))$ minimum spanning tree problems. A faster algorithm could be obtained by using a more sophisticated sorting algorithm. If a parallel algorithm which takes $O(log(|E|))$ rounds is used instead, then only $O(log^2(|E|))$ minimum spanning trees need to be solved. Unfortunately, this algorithm requires several cores to be implemented. An $O(log(|E|))$ factor could be saved by exploiting more the geometry of the problem. If all the intersection points of the edge composite costs $c_h + z w_h$ are known, then the optimal value z^* could be computed by performing a binary search over these points and solving $0(log(|E|))$ minimum spanning trees. However, computing all the intersection points requires $O(|E|^2)$ time. Instead of explicitly computing them, one could compute the median's

intersection point $\overline{z}$ and check if $\overline{z} < z^*$, $\overline{z} > z^*$ or $\overline{z} = z^*$. Note that from a geometrical point of view, determining the median intersection point is equivalent to finding the line associated with the edge composite costs $c_h + zw_h$ with the median slope [62]. An $O(|E|log(|E|))$ algorithm given in [18] could be used for this purpose. The total running time is $O(log(|E|)(|E|log(|E|) + t(MST)))$ time, where $t(MST)$ is the running time of a minimum spanning tree algorithm. In practice, it is sufficient to perform a binary search over an interval containing the unknown z^* value. The later is a breakpoint of the concave function defined by the intersection of two lines associated spanning trees $T^<_{z^*} = (V, F^<)$ and $T^>_{z^*} = (V, F^>)$. Therefore, z^* satisfies:

$$\sum_{e \in F^<} c_e + z^* w_e = \sum_{e \in F^>} c_e + z^* w_e, \tag{4.12}$$

which implies:

$$z^* = \frac{\sum_{e \in F^>} c_e - \sum_{e \in F^<} c_e}{\sum_{e \in F^<} w_e - \sum_{e \in F^>} w_e} < \sum_{e \in F^{maxc}} c_e - \sum_{e \in F^{minc}} c_e, \tag{4.13}$$

where $T^{maxc} = (V, F^{maxc})$ (respectively $T^{minc} = (V, F^{minc})$) is a spanning tree of G maximizing $\sum_{e \in F^{maxc}} c_e$ (respectively minimizing $\sum_{e \in F^{minc}} c_e$). In a first approximation, Algorithm 4 starts with interval $[lb_0; ub_0]$, such that $lb_0 = 0$ and $ub_0 = \sum_{e \in F^{maxc}} c_e - \sum_{e \in F^{minc}} c_e$, and reduces it iteratively. When its size reaches a predefined error precision ϵ , a procedure determines z^*. By choosing ϵ sufficiently small, the task will not be time consuming. In the extreme case, if ϵ is smaller than the distance separating two consecutive breakpoints, then the Lagrangian multiplier function has only one breakpoint over this tiny interval namely the one corresponding to z^*. By (4.13), this distance is at least

$$|z_1 - z_2| = |\frac{\sum_{e \in T^1} c_e - \sum_{e \in T^2} c_e}{\sum_{e \in T^2} w_e - \sum_{e \in T^1} w_e} - \frac{\sum_{e \in T^2} c_e - \sum_{e \in T^3} c_e}{\sum_{e \in T^3} w_e - \sum_{e \in T^2} w_e}| \tag{4.14}$$

$$|z_1 - z_2| > \frac{1}{(\sum_{e \in F^{maxw}} w_e - \sum_{e \in F^{mincw}} w_e)^2} \tag{4.15}$$

where $T^{maxw} = (V, F^{maxw})$ (respectively $T^{minw} = (V, F^{minw})$) is a spanning tree of G maximizing $\sum_{e \in F^{maxw}} w_e$ (respectively minimizing $\sum_{e \in F^{minw}} w_e$). There exist several known algorithms to compute the Lagrangian multiplier function over an interval $[lb; ub]$. Gusfield [36] and Yamada et al. [82] propose such algorithms.

Algorithm 4 is divided into two parts. First, a binary search exploits the concavity of $L(z)$ to find a small interval such that $z^* \in [lb; ub]$ (lines 1-2). Then the coordinate z^* is founded by exploring the breakpoints of $L(z)$ in $[lb; ub]$ and a feasible solution is found among all optimal solutions for $L(z^*)$ (lines 3-4).

Algorithm 4: Optimization procedure for the Lagrangian relaxation

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, $[lb; ub], lb, ub, \epsilon \in \mathbb{R}_+$

```
# binary search
lb, ub ← Algorithm 5(G, c, w, B, [lb; ub], ϵ);
# find z* and a tight feasible solution
z*, T>, T< ← Algorithm 6(G, c, w, B, [lb; ub]);
Return z*, T>, T<;
```

The binary search is developed in the following Algorithm 5:

Algorithm 5: Binary search

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, $[lb; ub], lb, ub, \epsilon \in \mathbb{R}_+$

```
1  while ub − lb > ϵ do
2      z ← (lb+ub)/2;
3      Compute the minimum lexicographic spanning tree T = (V, F) on
         (Σ_{e∈F} c_e + z w_e, Σ_{e∈F} −w_e);
4      if Σ_{e∈F} w_e < B then
5          ub ← z;
6      else
7          Compute the minimum lexicographic spanning tree T = (V, F) on
             (Σ_{e∈F} c_e + z w_e, Σ_{e∈F} w_e);
8          if Σ_{e∈F} w_e > B then
9              lb ← z;
10         else
11             Return z, z;
12         end
13     end
14 end
15 Return lb, ub;
```

Algorithm 5 proceeds binary search as long as the gap between ub and lb is higher than a given ϵ or until z^* is found during the procedure (line 11), in this case the interval $[z^*, z^*]$ is returned. At each loop, the research interval is reduced by half. The spanning tree T is computed using a lexicographic order, the latter considers the total composite cost and the total weight. Given two spanning trees $T_1 = (V, F_1)$ and $T_2 = (V, F_2)$, T_1 has a smaller value than T_2 $((\sum_{e\in F_1} c_e + zw_e, \sum_{e\in F_1} w_e) < (\sum_{e\in F_2} c_e + zw_e, \sum_{e\in F_2} w_e))$ if and only if $\sum_{e\in F_1} c_e + zw_e < \sum_{e\in F_2} c_e + zw_e$ or $\sum_{e\in F_1} c_e + zw_e = \sum_{e\in F_2} c_e + zw_e$ and $\sum_{e\in F_1} w_e < \sum_{e\in F_2} w_e$. The binary search of Algorithm 5 is illustrated in Figure 4.6, starting the search with $[lb_0; ub_0]$. Let lb_i (respectively ub_i) be the i-th value of the lower (respectively upper) bound of the research interval.

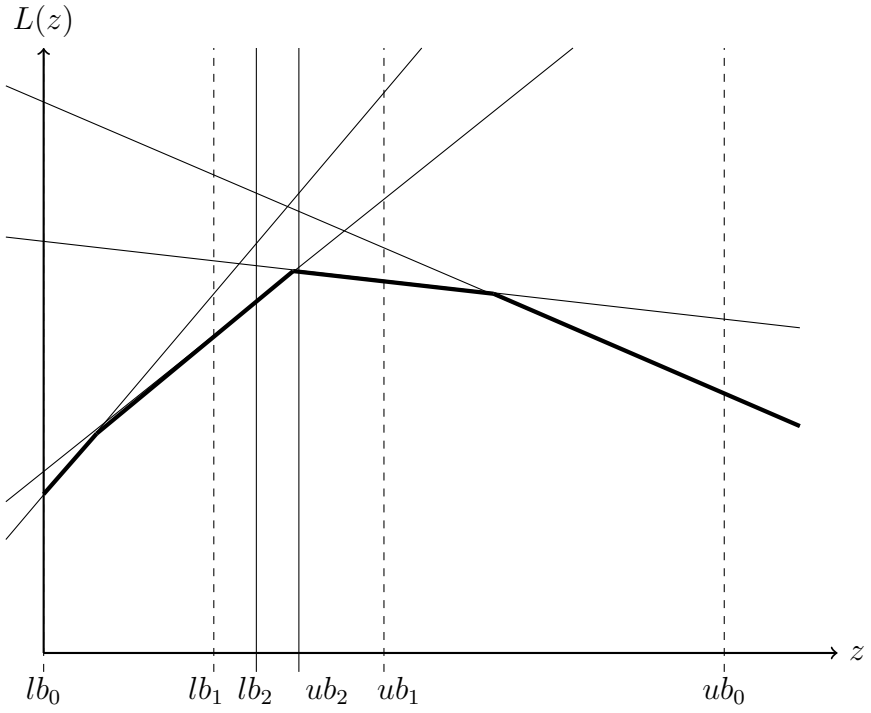


Figure 4.6: binary search of Algorithm 5. The lower envelope of the composite lines, corresponding to optimal solutions of $L(z)$, $z \geq 0$, is represented in bold. In the first iteration of the binary search, the upper bound is updated to ub_1. Then, in the second iteration the lower bound is updated to lb_1. In the third iteration, the search interval is $[lb_1; ub_2]$ and in the fourth iteration, the interval is updated to $[lb_2; ub_2]$.

After proceeding Algorithm 5, we have a restricted interval such that $z^* \in [lb; ub]$, it remains to find z^*.

Proposition 10. *Given $z' \in [lb; z^*[$, the closest z-coordinate of a breakpoint $z_{e^*f^*}$ in the piece-wise linear function $L(z)$ such that $z' < z_{e^*f^*}$ is given by $z_{e^*f^*} = \min\{\frac{c_f - c_e}{w_e - w_f} | e \in F, f \notin F, e$ in the cycle created by adding f to T, $\frac{c_f - c_e}{w_e - w_f} \geq z'\}$ where $T = (V, F)$ is a spanning tree minimizing $(\sum_{e \in F} c_e + z' w_e, \sum_{e \in F} w_e)$.*

Proof. By construction, T is a spanning tree with minimum total weight among all optimal spanning trees of $L(z')$. Hence, T is an optimal solution of $L(z)$ for $z \in [z'; z' + \epsilon]$, $0 < \epsilon << 1$. Assume that there exists z'', the minimum z-coordinate of a breakpoint of $L(z)$ such that $z' < z'' < z_{e^*f^*}$. Let $T'' = (V, F'')$ be the spanning tree such that the composite cost functions related to T and T'' intersect themselves at z''. By hypothesis T and T'' are both minimum spanning trees of $L(z'')$. By exploiting properties of minimum spanning trees, we can obtain T'' from T by performing a series of swaps of couple of edges with the same edge composite cost values at z''. Let $h \in F$ and $g \in F''$ be the first swap from T to T'', the edge composite cost functions related to h and g intersect themselves at z''. Moreover $z'' \geq \min\{\frac{c_f - c_e}{w_e - w_f} \mid e \in F, f \notin F,\ e$ in the cycle created by adding f to T, $\frac{c_f - c_e}{w_e - w_f} \geq z'\} = z_{e^*f^*}$, contradicting the minimality of z''. □

Figure 4.7 illustrates the relation between $z', z_{e^*f^*}$ and z''.

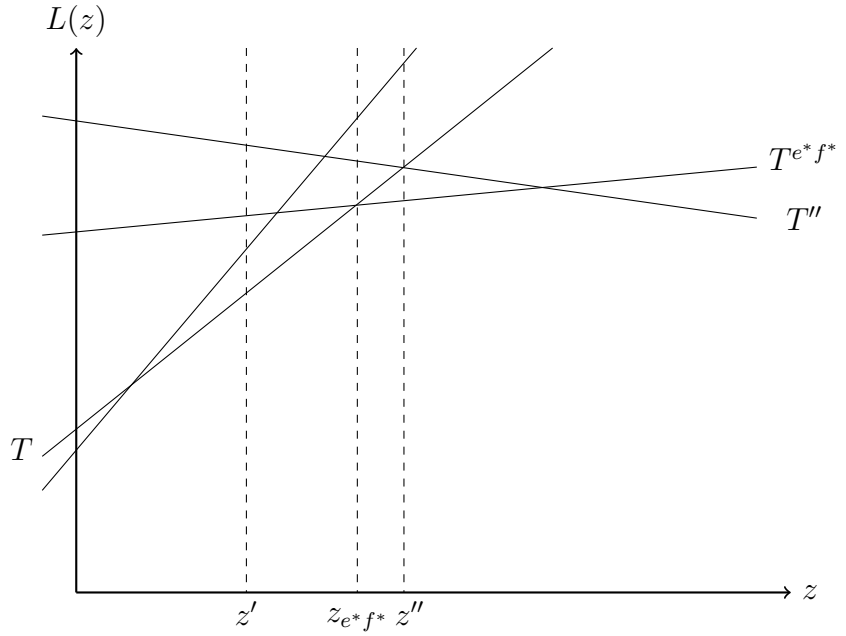


Figure 4.7: Composite cost functions of T and T''. The dashed lines associated with $z_{e^*f^*}$ and z'' represent the z-coordinates of the intersections of the related spanning tree composite cost functions with the composite cost function of T. T is an optimal solution of $L(z)$ for $z' \leq z \leq z_{e^*f^*}$ and $T^{e^*f^*}$ is an optimal solution of $L(z)$ for $z_{e^*f^*} \leq z \leq z^*$.

Notice that z^* corresponds to the first encountered breakpoint such that the spanning tree $T = (V, F)$ lexicographically minimizing $(\sum_{e \in F} c_e + zw_e, \sum_{e \in F} w_e)$ satisfies $\sum_{e \in F} w_e \leq B$. By successively iterating z-coordinates computed using Proposition 10, we are able to find z^*, as illustrated in Figure 4.8.

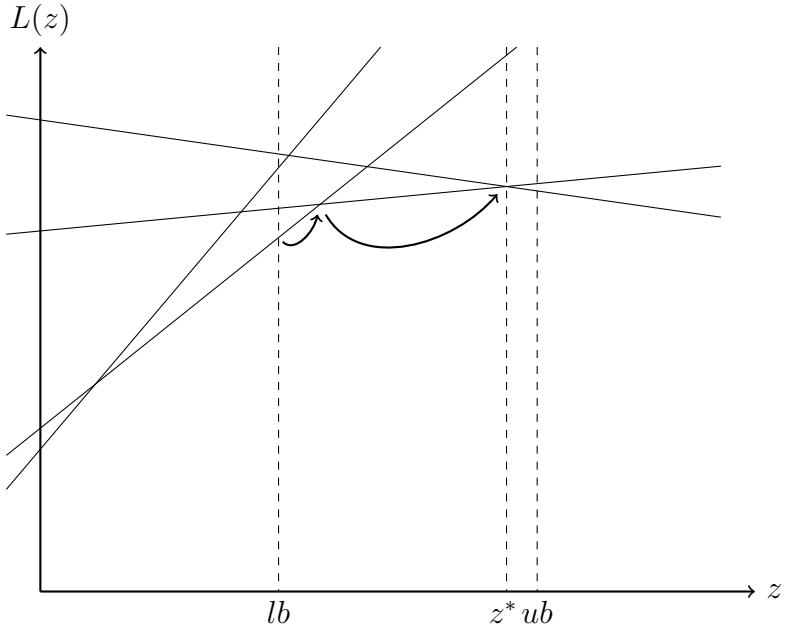


Figure 4.8: Iterating on successive breakpoints of $L(z)$ in $[lb; ub]$. The two arrows represents the two iterations that computes the successive breakpoints until converging to z^*.

Notice that z^* can be determined by iterating on successive breakpoints of $L(z)$ starting with $z = 0$. This way, z^* can be found by computing a minimum of one spanning tree. But this procedure can be time consuming in practice as computing the next breakpoint requires $O(|V||E|)$ operations. In Algorithm 6, this procedure is done only few times as the interval $[lb; ub]$, obtained after the binary search, is small.

There are at least two optimal solutions of $L(z^*)$, a spanning tree $T^{z^*+\epsilon}$ satisfying the budget constraint and a spanning tree $T^{z^*-\epsilon}$ that violates the budget constraint. Furthermore, there may be a high number of optimal solutions of $L(z^*)$ as illustrated in Figure 4.9. In the following, we give a heuristic that computes among the feasible optimal solutions of $L(z^*)$, a solution with a small total cost.

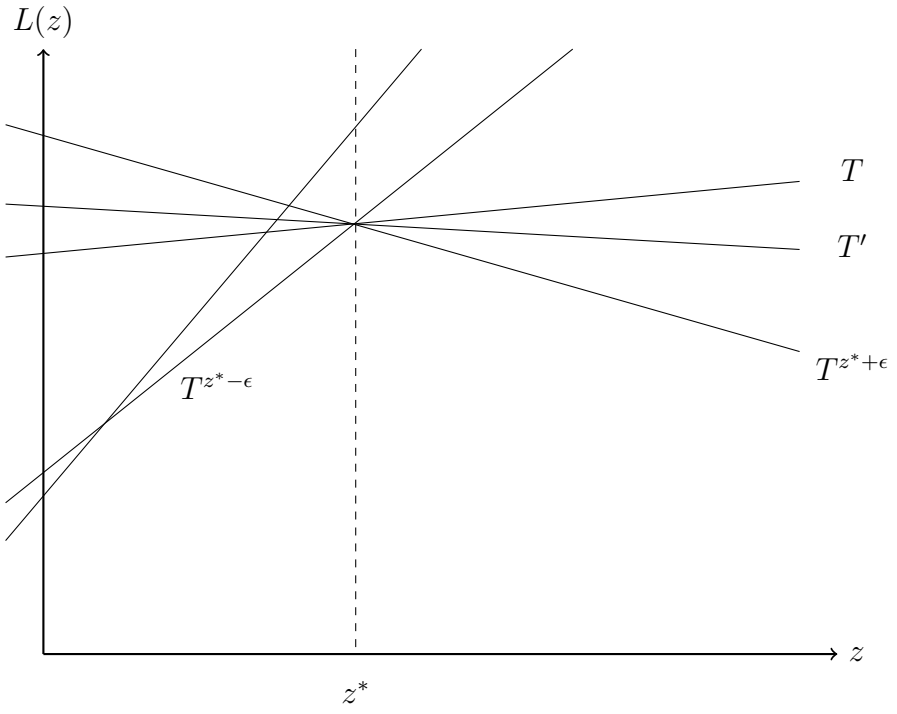


Figure 4.9: Several composite costs functions intersecting at $(z^*, L(z^*))$. Spanning trees $T, T', T^{z^*-\epsilon}$ and $T^{z^*+\epsilon}$ are optimal solutions of $L(z^*)$. T and $T^{z^*-\epsilon}$ are infeasible solutions and T' and $T^{z^*+\epsilon}$ are feasible solutions. Among both feasible solutions optimal at $L(z^*)$, T' minimizes the total cost.

Algorithm 6 is called after the binary search Algorithm 4, it finds the coordinate z^* iterating on successive breakpoints (lines 2-7) and computes a feasible solution with a small total cost among all the feasible optimal solutions at $L(z^*)$ (lines 9-15). In the latter procedure, starting with $T^{z^*-\epsilon}$ we heuristically perform swaps of edges with the same edge composite cost values at z^* such that at each iteration the decrease in total weight is minimum.

Algorithm 6: Computation of z^* and a tight feasible solution

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, $[lb; ub], lb, ub \in \mathbb{R}_+$

1 # **iteratively compute the next breakpoint until finding** z^*
2 Compute a minimum lexicographic spanning tree $T = (V, F)$ on $(\sum_{e \in F} c_e + lbw_e, \sum_{e \in F} w_e)$;
3 **while** $\sum_{e \in F} w_e > B$ **do**
4 $\quad e^*, f^* \leftarrow \arg\min\{\frac{c_f - c_e}{w_e - w_f} \mid e \in F, f \notin F, e$ in the cycle created by adding f to $T, \frac{c_f - c_e}{w_e - w_f} \geq lb\}$;
5 $\quad lb \leftarrow \frac{c_{f^*} - c_{e^*}}{w_{e^*} - w_{f^*}}$;
6 $\quad$ Compute a minimum lexicographic spanning tree $T = (V, F)$ on $(\sum_{e \in F} c_e + lbw_e, \sum_{e \in F} w_e)$;
7 **end**
8 # **computing a tight feasible solution**
9 Compute a minimum lexicographic spanning tree $T^< = (V, F^<)$ on $(\sum_{e \in F^<} c_e + lbw_e, \sum_{e \in F^<} -w_e)$;
10 $T^> = (V, F^>) \leftarrow (V, \emptyset)$;
11 **while** $\sum_{e \in F^<} w_e > B$ **do**
12 $\quad e^*, f^* \leftarrow \arg\min\{w_e - w_f \mid e \in F^<, f \notin F^<, e$ in the cycle created by adding f to $T^<, \frac{c_f - c_e}{w_e - w_f} = lb, w_e - w_f > 0\}$;
13 $\quad F^> \leftarrow F^<$;
14 $\quad F^< \leftarrow F^< \cup \{f^*\} \backslash \{e^*\}$;
15 **end**
16 Return $lb, T^>, T^<$;

Algorithm 4 has a complexity of $O(log(|E|)|E|log(|V|) + |E|^3|V|)$. The $|E|^3$ factor corresponds to the upper bound $|E|^2$ on the number of breakpoints in $L(z)$. In practice as Algorithm 5 reduces the search interval, we require few swaps to converge to z^*.

In the following, we introduce a preprocessing procedure allowing one to reduce the resolution time of Algorithm 4. The latter computes several minimum spanning trees. Although those spanning trees can be solved in $O(|E|log(|V|))$, several calls may be time consuming. By exploiting the geometry of the edge's composite cost functions, we can reduce the size of the instance and then the computational time. Let $lb < z^*$, $ub > z^*$, and $T_{lb} = (V, E_{lb})$ (respectively $T_{ub} = (V, E_{ub})$) be a minimum spanning tree with respect to $L(lb)$ (respectively $L(ub)$). Let $z' = \frac{\sum_{e \in E_{lb}} c_e - \sum_{e \in E_{ub}} c_e}{\sum_{e \in E_{ub}} w_e - \sum_{e \in E_{lb}} w_e}$ be the z-coordinate of the intersection of the linear cost composite functions associated with T_{lb} and T_{ub}.

Proposition 11. *Any edge $e \in E$ satisfying the three conditions:*

- $c_e + z'w_e > \max\{c_f + z'w_f \mid f \in E_{lb} \cup E_{ub}\}$,
- $c_e + lbw_e > \max\{c_f + lbw_f \mid f \in E_{lb}\}$,
- $c_e + ubw_e > \max\{c_f + ubw_f \mid f \in E_{ub}\}$,

cannot belongs to any optimal solution of the Lagrangian relaxation.

Proof. Assume that an edge e satisfying the three conditions belongs to an optimal solution T^* of the Lagrangian relaxation. First, if $z^* \in [lb; z']$, by hypothesis, $c_e + zw_e > c_f + zw_f, \forall f \in E_{lb}, \forall z \in [lb; z']$, then by swapping e from T^* with an edge in the induced cut and in E_{ub} we obtain a spanning tree with a smaller objective value than T^*, a contradiction. Now, if $z^* \in [z'; ub]$. By hypothesis, $c_e + zw_e > c_f + zw_f, \forall f \in E_{ub}, \forall z \in [z'; ub]$, then by swapping e from T^* with an edge in the induced cut and in E_{ub} we obtain a spanning tree with a smaller objective value than T^*, a contradiction. □

Figure 4.10 illustrates the proof of Proposition 11.

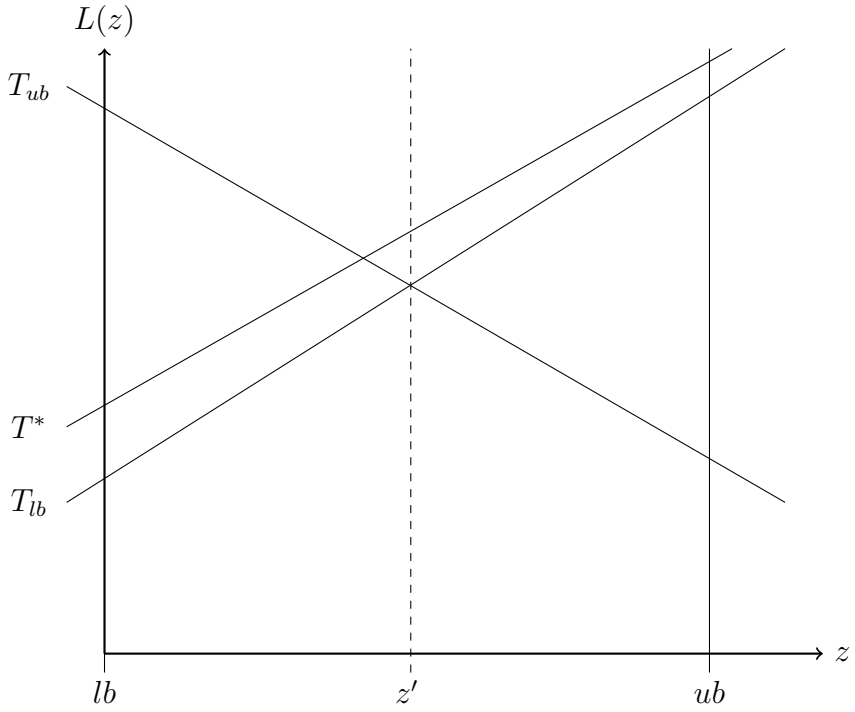


Figure 4.10: Composite costs of T_{lb}, T_{ub} and T^*.

We deduce the following algorithm from Proposition 11:

Algorithm 7: Preprocessing - Deletion of edges that cannot belong to a optimal solution of the Lagrangian relaxation

Data: Graph $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{|E|}$, $[lb; ub], lb, ub \in \mathbb{R}_+$

Let $T_{lb} = (V, E_{lb})$ be the lexicographically minimum spanning tree with respect to $(\sum_{e \in E_{lb}} c_e + lbw_e, \sum_{e \in E_{lb}} w_e)$;

Let $T_{ub} = (V, E_{ub})$ be the lexicographically minimum spanning tree with respect to $(\sum_{e \in E_{ub}} c_e + ubw_e, \sum_{e \in E_{ub}} -w_e)$;

Let $z' = \frac{\sum_{e \in E_{lb}} c_e - \sum_{e \in E_{ub}} c_e}{\sum_{e \in E_{ub}} w_e - \sum_{e \in E_{lb}} w_e}$;

Let $y_{z'} = \max\{c_e + z'w_e \mid e \in E_{lb} \cup E_{ub}\}$;

Let $y_{lb} = \max\{c_e + lbw_e \mid e \in E_{lb}\}$;

Let $y_{ub} = \max\{c_e + ubw_e \mid e \in E_{ub}\}$;

for $e \in E \backslash (E_{lb} \cup E_{ub})$ **do**

  if $c_e + z'w_e > y_{z'}$ and $c_e + lbw_e > y_{lb}$ and $c_e + ubw_e > y_{ub}$ **then**

    Remove e from E

  end

end

Algorithm 7 can be called multiple times during the binary phase. In the following, we run Algorithm 7 a first time when $ub - lb < 1$ and a second time when $ub - lb < 10^{-3}$ during the binary search part. Table 4.1 records the running time of the cutting plane method for solving the linear program (4.1)-(4.5) and our implementations of Megiddo's approach on random dense graphs with strongly uncorrelated costs and weights.

$\|V\|$	Resolution Time (s) Separation Alg. 2	Resolution Time (s) Algorithm 4	Resolution Time (s) Algorithm 4 + 7
100	0.348	0.021	0.017
200	6.767	0.112	0.088
300	60.082	1.023	0.347
400	228.420	2.731	0.819
500	657.672	4.235	1.217
600	3600.000	6.014	1.893
700	3600.000	9.002	2.782
800	3600.000	11.315	3.967
900	3600.000	14.539	4.852
1000	3600.000	18.223	5.671
1500	3600.000	40.003	13.044
2000	3600.000	73.440	23.180

Table 4.1: Time comparison between the linear and the Lagrangian relaxations

These computational experiments show that the Lagrangian relaxation technique is much faster than the cutting plane method. Furthermore, the preprocessing improves the running time. The results of the linear relaxation can be explained by the high number of calls to the subtour separation algorithm. After many separation iterations, the linear

program contains a huge number of subtour constraints and becomes time consuming to solve. The resolution of Algorithm 4 can be improved by exploiting the structure of the data, starting with a small input search interval $[lb; ub]$ and a preprocessing procedure. Moreover, we used Prim's algorithm for all minimum spanning tree computations as it appears to be the most effective in practice.

4.2.2 Special case of the Lagrangian relaxation

After focusing on the linear relaxation in the general case, we consider the special case such that every edge $e \in E$ has a binary cost $c_e \in \{0, 1\}$. In this special case, rounding the linear relaxation leads to an optimal solution for the integer problem, see Corollary 1. The latter can then be solved in polynomial time. Similarly to the previous sections, we exploit the Lagrangian relaxation in order to efficiently compute the linear relaxation. Notice that we consider the maximization version of the problem as we solve such problems in following chapters. Consider the following linear program:

$$\max \sum_{e \in E} c_e x_e \tag{4.16}$$

$$s.t. \sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subset V, 2 < |S| < |V| - 1, \tag{4.17}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{4.18}$$

$$\sum_{e \in E} w_e x_e \leq B, \tag{4.19}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{4.20}$$

Similarly to the general case, by dualizing the budget constraint (4.19), we obtain the following problem:

$$\max_{z \geq 0} \ \min \sum_{e \in E} -c_e x_e - z(B - \sum_{e \in E} w_e x_e) \tag{4.21}$$

$$\sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subset V, 2 < |S| < |V| - 1, \tag{4.22}$$

$$(LR) \qquad \sum_{e \in E} x_e = |V| - 1, \tag{4.23}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{4.24}$$

We solve problem (4.21)-(4.24) using geometry. The edge set E can be partitioned into two sets E^0 and E^1, such that $E^i = \{e \in E \mid c_e = i\}$, $i = 0, 1$. By construction, all composite cost functions of edges belonging to E^0 (respectively E^1) intersect themselves at $(0, 0)$ (respectively at $(0, -1)$). Moreover, every intersection of two composite cost functions at $z > 0$ implies an edge in E^0 and an edge in E^1. Figure 4.11 illustrates the geometry of this special case.

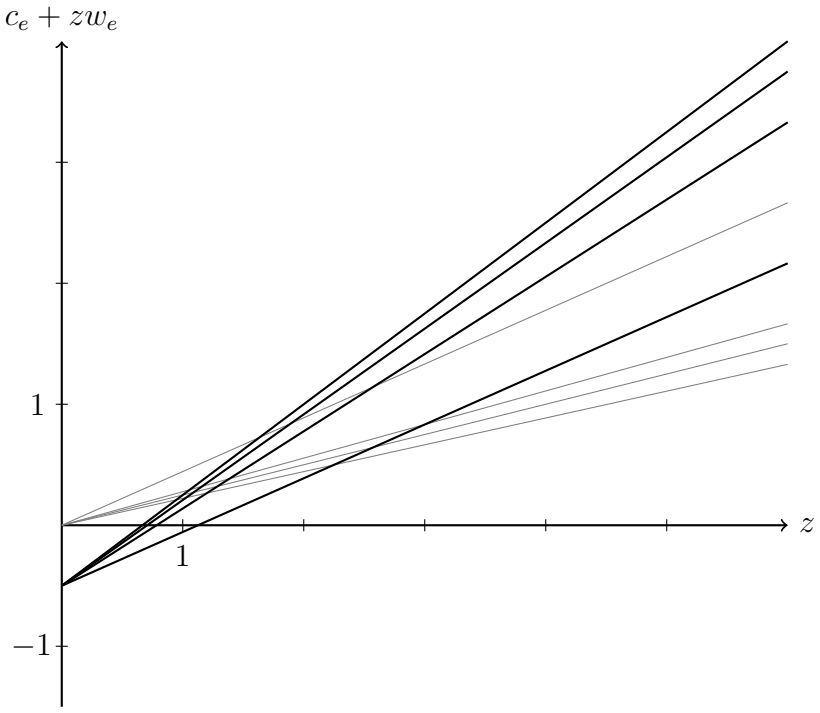


Figure 4.11: Edge composite costs functions of the special case where the costs are binary. Composite cost functions of edges in E^0 are represented in gray and functions of edges in E^1 are represented in black.

The particular geometry of this special case can be exploited to reduce the instance size in a preprocessing phase. Let G^0 and G^1 be subgraphs of G such that $G^i = (V, E^i),\ i = 0, 1$ and let $F^i = (V, H^i),\ i = 0, 1$ be forests of G^i minimizing lexicographically $(-|H^i|,\ \sum_{e \in H^i} w_e)$.

Proposition 12. *There exists an optimal solution $T = (V, H)$ of (4.21)-(4.24) such that $H \subseteq H^0 \cup H^1$.*

Proof. Assume that there is no optimal solution of (4.21)-(4.24) with an edge set in $H^0 \cup H^1$. Consider an optimal solution $T^* = (V, H^*)$ of (4.21)-(4.24), T^* contains at least one edge in $E \backslash (H^0 \cup H^1)$. Consider any edge $e \in H^* \backslash (H^0 \cup H^1)$, by adding e to F^{c_e} we create a cycle C (by minimality of $-|H^{c_e}|$) such that $w_e \geq w_f,\ \forall f \in C \backslash \{e\}$ (by minimality of $\sum_{e \in H^{c_e}} w_e$). As $c_f = c_e$, we deduce that $c_e + zw_e \geq c_f + zw_f,\ \forall z \geq 0$. By swapping e with any edge in $C \backslash \{e\}$ and in the cut induced by deleting e from T^*, we obtain a spanning tree T^1 such that $c(T^1) + zw(T^1) \leq c(T^*) + zw(T^*),\ z \geq 0$. By repeating this operation for every remaining edge in $H^* \backslash (H^0 \cup H^1 \cup \{e\})$, we obtain a feasible optimal solution T with only edges in $H^0 \cup H^1$ and such that $c(T) + zw(T) \leq c(T^{|H^* \backslash (H^0 \cup H^1)| - 1}) + zw(T^{|H^* \backslash (H^0 \cup H^1)| - 1}) \leq \cdots \leq c(T^1) + zw(T^1) \leq c(T^*) + zw(T^*),\ z \geq 0$, a contradiction. □

In the following, we adapt Algorithm 4 in order to efficiently solve the Lagrangian relaxation (4.21)-(4.24). Algorithm 8 is divided into three parts. First we run a preprocessing phase (lines 2-6) in order to reduce the size of the problem to at most $(2|V| - 2)$ edges. Then we perform a binary search (line 8) to reduce the search interval $[lb; ub]$ containing

z^* and we compute the best objective value of (4.16)-(4.20) (line 10).

Algorithm 8: Optimization procedure for problem (4.21)-(4.24)

Data: Graph $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, $[lb; ub], lb, ub, \epsilon \in \mathbb{R}_+$
1 # **preprocessing**
2 $S \leftarrow H^0 \cup H^1$;
3 Compute a spanning tree $T = (V, F)$ minimizing $\sum_{e \in F} -c_e$ on $G' = (V, S)$;
4 **if** $\sum_{e \in F} w_e \leq B$ **then**
5 | Return $|F \cap H^1|$;
6 **end**
7 # **binary search**
8 $lb, ub \leftarrow$ Algorithm 9 $(G' = (V, S), c, w, B, [lb; ub], \epsilon)$;
9 # **finding an optimal solution**
10 $sol \leftarrow$ Algorithm 10$(G' = (V, S), H^0, H^1, c, w, B, [lb; ub])$;
11 Return sol;

Algorithm 9 adapts the binary search Algorithm 5 to the special case.

Algorithm 9: Binary search

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, $[lb; ub], lb, ub, \epsilon \in \mathbb{R}_+$
1 **while** $ub - lb > \epsilon$ **do**
2 | $z \leftarrow \frac{lb+ub}{2}$;
3 | Compute the minimum lexicographic spanning tree $T^> = (V, F^>)$ on $(\sum_{e \in F^>} -c_e + zw_e, \sum_{e \in F^>} -w_e)$;
4 | **if** $\sum_{e \in F^>} w_e < B$ **then**
5 | | $ub \leftarrow z$;
6 | **else**
7 | | Compute the minimum lexicographic spanning tree $T^< = (V, F^<)$ on $(\sum_{e \in F^<} -c_e + zw_e, \sum_{e \in F^<} w_e)$;
8 | | **if** $\sum_{e \in F^<} w_e > B$ **then**
9 | | | $lb \leftarrow z$;
10 | | **else**
11 | | | Return z, z;
12 | | **end**
13 | **end**
14 **end**
15 Return lb, ub;

Algorithm 10 adapts Algorithm 6. First, z^* is found by iterating on successive breakpoints. Then an optimal solution to the integer problem (4.16)-(4.20) is computed by performing swaps. In the binary case, every swap at z^* between an edge in $e \in H_1$ and an edge $f \in H_0$ decreases the total weight of the solution by the same value $w_e - w_f$. By performing swaps until we get a feasible solution, we obtain an optimal solution for the

integer version of the problem (4.16)-(4.20).

Algorithm 10: Computation of z^* and an integer optimal solution

Data: Graph $G = (V, E), H^0, H^1 \subseteq E$, $c \in \mathbb{R}_+^{|E|}, w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, $[lb; ub], lb, ub \in \mathbb{R}_+$

1 # **iteratively compute the next breakpoint until finding** z^*
2 Compute the minimum lexicographic spanning tree $T = (V, F)$ on $(\sum_{e\in F} -c_e + lbw_e, \sum_{e\in F} w_e)$;
3 **while** $\sum_{e\in F} w_e > B$ **do**
4 $\quad e^*, f^* \leftarrow \arg\max\{w_f - w_e \mid e \in F \cap H^1, f \in H^0\backslash F, e$ in the cycle created by adding f to $T, \frac{1}{w_f - w_e} \geq lb\}$;
5 $\quad lb \leftarrow \frac{1}{w_{f^*} - w_{e^*}}$;
6 $\quad$ Compute the minimum lexicographic spanning tree $T = (V, F)$ on $(\sum_{e\in F} -c_e + lbw_e, \sum_{e\in F} w_e)$;
7 **end**
8 # **computing an optimal solution**
9 Compute the minimum lexicographic spanning tree $T^< = (V, F^<)$ on $(\sum_{e\in F^<} -c_e + lbw_e, \sum_{e\in F^<} -w_e)$;
10 $T^> = (V, F^>) \leftarrow (V, \emptyset)$;
11 **while** $\sum_{e\in F^<} w_e > B$ **do**
12 $\quad$ Let $e \in F^< \cap H^1$ and $f \in H^0\backslash F^<$ such that e belongs to the cycle created by adding f to $T^<$ and $\frac{1}{w_f - w_e} = lb$;
13 $\quad F^> \leftarrow F^<$;
14 $\quad F^< \leftarrow F^< \cup \{f\}\backslash\{e\}$;
15 **end**
16 Return $|H^1 \cap F^<|$;

The following proposition, used in Algorithm 10, adapts Proposition 10 for the binary case.

Proposition 13. *Given $z' \in [lb; z^*[$, the closest breakpoint $z_{e^*f^*}$ in the piece-wise linear function $L(z)$ such that $z' < z_{e^*f^*}$ is given by $z_{e^*f^*} = \min\{\frac{1}{w_f - w_e} \mid e \in F \cap H_1, f \in H_0\backslash F, e$ in the cycle created by adding f to $T, \frac{1}{w_f - w_e} \geq z'\}$ where $T = (V, F)$ is a spanning tree minimizing $(\sum_{e\in F} c_e + z'w_e, \sum_{e\in F} w_e)$.*

Proof. By construction, T is a spanning tree with a minimum total weight among all optimal spanning trees for $L(z')$. Then T is an optimal solution of $L(z)$ for $z \in [z'; z' + \epsilon]$, $0 < \epsilon << 1$. Assume that z'' is the minimum z-coordinate of a breakpoint of $L(z)$ such that $z' < z'' < z_{e^*f^*}$. Let $T'' = (V, F'')$ be the spanning tree such that the cost composite functions related to T and T'' intersect at z''. By hypothesis T and T'' are both minimum solutions of $L(z'')$. By exploiting properties of minimum spanning trees, we can obtain T'' from T by performing a series of swaps of couple of edges with the same edge composite costs at z''. Let $h \in F \cap H^1$ and $g \in F'' \cap H^0$ be the first swap from T to T'', the edge composite cost functions related to h and g intersect at z''. Hence $z'' \geq \min\{\frac{1}{w_f - w_e} \mid e \in F \cap H_1, f \in H_0\backslash F, e$ in the cycle created by adding f to $T, \frac{1}{w_f - w_e} \geq$

$z'\} = z_{e^*f^*}$, contradicting the minimality of z''. □

Chapter 5

Extended formulations for the 1-budgeted spanning tree problem

We propose in this chapter an exact extended formulation that builds upon the work of Hassin and Levin [37]. The authors designed an approximation algorithm for the M1BSTP using Lagrangian relaxation technique, a linear characterization of the spanning tree polytope, and matroid intersection. Depending on the error precision, the algorithm partitions the set of edges E into subsets $E_1, \ldots, E_k$ and computes a number of spanning trees with n_i edges from set E_i for $i = 1, \ldots, k$. This task is performed by solving matroid intersection problems: the graphic matroid and the partition matroid. Extended formulations are important in integer programming in both theory and practice. Martin et al. [50] propose an exact extended formulation for the knapsack problem (KP) based on the dynamic programming algorithm. By adding the spanning tree basic inequalities, Martin et al.'s formulation can be generalized to our problem. However, its projection yields loose valid inequalities.

In contrast to Martin et al. [50] formulation, our formulation is exponentially large that is prohibitive to solve. However, it allows us to gain some insights on the structural properties of the problem. Due to the spanning tree constraints, our problem seems to be more difficult than the KP. Surprisingly, our formulation shows that, like the KP, the difficulty of the problem lies also in the number of distinct coefficient values in the budget constraint and not the magnitude of formulation's parameters. Using dynamic programming, the KP can be solved in polynomial time if the number of distinct coefficients in the constraint is a constant.

In the particular case where the graph is a cactus, we also propose an exact extended formulation for the dBSTP based on dynamic programming. The size of this formulation is pseudo-polynomially bounded.

5.1 Extended formulation based on matroid intersection

Before introducing our integer extended formulation, we will first define the required substructures. Consider a partition of the edge set E into the sets $E_i = \{e \in E \mid w_e = w_i\},\ i = 1, \ldots, k$ where $w_1, \ldots w_k$ denote the distinct weight values among all the edges in E. Our extended formulation is based on enumerating all the possible integer vectors

$h = (h^1, \ldots, h^k) \in \mathbb{N}^k$ satisfying:

$$\sum_{i=1}^{k} h^i = |V| - 1, \tag{5.1}$$

$$h^i \leq |E_i|, \ \forall i = 1, \ldots, k, \tag{5.2}$$

$$\sum_{i=1}^{k} w_i h^i \leq B. \tag{5.3}$$

For each such vector $h = (h^1, \ldots, h^k)$, consider the following matroid intersection polyhedron P_h:

$$\sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subseteq V, 2 \leq |S|, \tag{5.4}$$

$$\sum_{e \in E_i} x_e \leq h^i, \qquad i = 1, \ldots, k, \tag{5.5}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{5.6}$$

Constraints (5.4) and (5.6) describe the graphical matroid polytope, and constraints (5.5) and (5.6) correspond to a partition matroid. By Edmond's theorem [24], the polyhedron P_h is integral. For any vector h satisfying (5.1)-(5.3), the set of extreme points of P_h coincides with the set of forests of G satisfying constraints (5.5) related to h. In order to focus on spanning tree solutions, we consider the polyhedron $\mathcal{P}_h = \{x \in \mathbb{R}^E \mid x \in P_h, \ \sum_{e \in E} x_e = |V| - 1\}$.

Proposition 14. *$\mathcal{P}_h$ is integral.*

Proof. $\mathcal{P}_h$ is a proper face of the integral polyhedron P_h, then it is also integral. □

Let $Q^* = \{h = (h^1, \ldots, h^k)$ satisfying $(5.1) - (5.3) \mid \mathcal{P}_h \neq \emptyset\}$. Q^* contains an exponentially large number of vectors. The set of feasible solutions of the 1BSTP, denoted by $\mathcal{H}$, lies in the union of polyhedra $\bigcup_{h \in Q^*} \mathcal{P}_h$.

Example 2

Consider the graph $G = (V, E)$ given by Figure 5.1, with weights on the edges and a budget of 5.

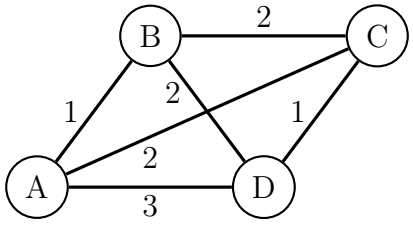


Figure 5.1: An instance of the 1BSTP.

By partitioning the set of edges according to their weights, we obtain 3 sets: $E_1 = \{AB, CD\}$, $E_2 = \{AC, BC, BD\}$ and $E_3 = \{AD\}$. Enumerating all possible vectors $h = (h^1, h^2, h^3) \in \mathbb{N}^3$ belonging to Q^* we obtain the three vectors $(2, 1, 0)$, $(2, 0, 1)$ and $(1, 2, 0)$. The associated polyhedron $\mathcal{P}_{(2,1,0)}$ is given by:

$$
\begin{aligned}
& x_{AB} + x_{AD} + x_{BD} \leq 2, \\
& x_{BC} + x_{BD} + x_{CD} \leq 2, \\
& x_{AB} + x_{AC} + x_{BC} \leq 2, \\
& x_{AC} + x_{AD} + x_{CD} \leq 2, \\
& x_{AB} + x_{AC} + x_{AD} + x_{BC} + x_{BD} + x_{CD} = 3, \\
& x_{AB} + x_{CD} \leq 2, \\
& x_{AC} + x_{BC} + x_{BD} \leq 1, \\
& x_{AD} \leq 0, \\
& 0 \leq x_{AB}, x_{AC}, x_{AD}, x_{BC}, x_{BD}, x_{CD} \leq 1.
\end{aligned}
$$

$\mathcal{P}_{(2,1,0)}$ is integral and its extreme points correspond to the set of spanning trees in G satisfying the matroid partition constraints (5.5) related to $(2, 1, 0)$.

Balas [8] provides an extended formulation to model the union of polyhedra $\bigcup_{h \in Q^*} \mathcal{P}_h$. In the latter formulation, every polyhedron $\mathcal{P}_h$ is associated with variables y_h and with a binary variable y_0^h indicating whether the vector x belongs to polyhedron $\mathcal{P}_h$. Note that variables y^h denote a copy of the original vector of design variables. Let $Ay^h \leq b^h$ corresponds to the set of constraints related to (5.4)-(5.6). Notice that the left-hand matrix A is the same for every $\mathcal{P}_h$, $h \in Q^*$. Balas [8] shows that $x \in \mathbb{R}^E$ belongs to the union of polyhedra $\mathcal{P}_h$ for $h \in Q^*$ if and only if

$$
\begin{aligned}
& x_e - \sum_{h \in Q^*} y_e^h = 0, && \forall e \in E, && (5.7) \\
& \sum_{h \in Q^*} y_0^h = 1, && && (5.8) \\
& Ay^h - b^h y_0^h \leq 0, && \forall h \in Q^*, && (5.9) \\
& \sum_{e \in E} y_e^h - (|V| - 1) y_0^h = 0, && \forall h \in Q^*, && (5.10) \\
& y_0^h \in \{0, 1\}, && \forall h \in Q^*. && (5.11)
\end{aligned}
$$

Balas [8] showed that the convex hull of the solutions of $(5.7) - (5.11)$ is obtained by relaxing the integrality constraints of variables y_0^h.

$$x_e - \sum_{h \in Q^*} y_e^h = 0, \qquad \forall e \in E, \tag{5.12}$$

$$\sum_{h \in Q^*} y_0^h = 1, \tag{5.13}$$

$$Ay^h - b^h y_0^h \leq 0, \qquad \forall h \in Q^*, \tag{5.14}$$

$$\sum_{e \in E} y_e^h - (|V| - 1) y_0^h = 0, \qquad \forall h \in Q^*, \tag{5.15}$$

$$y_0^h \geq 0, \qquad \forall h \in Q^*. \tag{5.16}$$

Since polyhedra $\mathcal{P}_h$ in the disjunction are integral, by Edmond's theorem [24] the above formulation is hence integral.

Proposition 15. *conv*$(\mathcal{H}) = \{x \in \mathbb{R}^{|E|}$ such that there exists $(y^h, y_0^h) \in \mathbb{R}^{(|E|+1)|Q^*|}$ where (x, y^h, y_0^h) is a solution of (5.12) − (5.16)$\}$.

The next result presents an extended formulation obtained by removing the cardinality constraint from the disjunction part. It provides a structural property satisfied by all the facets of the convex hull of the solutions in $\mathcal{H}$.

$$\min \sum_{e \in E} c_e x_e \tag{5.17}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{5.18}$$

$$x_e - \sum_{h \in Q^*} y_e^h = 0, \qquad \forall e \in E, \tag{5.19}$$

$$\sum_{h \in Q^*} y_0^h = 1, \tag{5.20}$$

$$Ay^h - b^h y_0^h \leq 0, \qquad \forall h \in Q^*, \tag{5.21}$$

$$y_0^h \geq 0, \qquad \forall h \in Q^*. \tag{5.22}$$

Proposition 16. *conv*$(\mathcal{H}) = \{x \in \mathbb{R}^{|E|}$ such that there exists $(y^h, y_0^h) \in \mathbb{R}^{(|E|+1)|Q^*|}$ where (x, y^h, y_0^h) is a solution of (5.18) − (5.22)$\} = BSTP^*(G)$.

Proof. We first show that *conv*$(\mathcal{H}) \subseteq BSTP^*(G)$. Any $x^* \in$ *conv*$(\mathcal{H})$ is a convex combination of a finite number of incidence vectors $x^1, \ldots, x^l$ of feasible spanning trees. By construction, every vector $x^i, i = 1, \ldots, l$ belongs to at least one polytope $\{x \in \mathbb{R}^{|E|} \mid Ax \leq b^h\}$, $h \in Q^*$. x^* can then be written as a convex combinations of at most $|Q^*|$ points p_h, each from a different $\{x \in \mathbb{R}^{|E|} \mid Ax \leq b^h\}$, $h \in Q^*$ (if several incidence vectors belong to the same polytope, p_h corresponds to their own convex combination). Each point p_h satisfies the cardinality constraint (5.18). We then have $x = \sum_{h \in Q^*} \lambda_h p_h$, where scalars $\lambda_h \geq 0, \forall h \in Q^*$ satisfy $\sum_{h \in Q^*} \lambda_h = 1$. For each point $p_h, h \in Q^*$, we can find the associated vector (x', y^i, y_0^i) with $x' = p_h$, $y^i = p_h$ for $i = h$, $y^i = 0$ for $i \neq h$, $y_0^i = 1$ for $i = h$, and $y_0^i = 0$ for $i \neq h$. The point x^* can be associated with the vector (x', y^i, y_0^i), where

$x' = x^*$, $y^h = p_h\lambda_h$ and $y_0^i = \lambda_h, \forall h \in Q^*$. All the vectors satisfy the constraints of the disjunctive formulation (5.19)-(5.22) and the cardinality constraint (5.18). This proves the first part.

Now assume that there exists a solution (x^*, y^*, y_0^*) in the polyhedron (5.18)-(5.22) such that $x^* \notin conv(\mathcal{H})$. (x^*, y^*, y_0^*) is a convex combination of extreme points (x^1, y^1, y_0^1), $\ldots, (x^l, y^l, y_0^l)$ of (5.18)-(5.22). Note that (5.19)-(5.22) correspond to the disjunction of polyhedra $\{y^h \in \mathbb{R}^{|E|} \mid Ay^h \leq b^h\},\ h \in Q^*$, where $\{y^h \in \mathbb{R}^{|E|} \mid Ay^h \leq b^h\},\ h \in Q^*$ is the intersection of both the graphic matroid and the partition matroid related to h. Therefore polyhedron (5.19)-(5.22) is integral by Balas [8] and every extreme point is associated with a forest that satisfies the budget constraint. Since the polyhedron (5.18)-(5.22) is a proper face of polyhedron (5.19)-(5.22), (5.18)-(5.22) is integral and $(x^1, y^1, y_0^1), \ldots, (x^l, y^l, y_0^l)$ are extreme points. Therefore, the set of solutions $x^1, \ldots, x^l$ are spanning trees that satisfy the budget constraint. Consequently, $x^1, \ldots, x^l$ belong to $\mathcal{H}$ and $x^* \in conv(\mathcal{H})$. This leads to a contradiction and proves the second part.

□

Let $\mathcal{P}'$ be the polyhedron given by (5.18)-(5.22). The constraints matrix of the extended formulation is almost block diagonal as illustrated in Figure 5.2. The first three rows represent constraints (5.18), (5.19) and (5.20), respectively. Note that by construction, only constraints (5.18) and (5.20) have a non-zero right-hand side. Each polyhedron $P_h, h \in Q^*$ in the disjunction is associated with a block corresponding to constraints (5.21) and (5.22). Let l be the number of rows of the matrix A (l is exponentially large due to the subtour constraint).

$$
\begin{pmatrix}
\mathbf{1_{1,|E|}} & 0_{1,|E|} & \cdots & & \cdots & & \cdots & 0_{1,|E|} & 0 \\
\boldsymbol{Id_{|E|}} & \boldsymbol{-Id_{|E|}} & 0_{|E|,1} & \boldsymbol{-Id_{|E|}} & 0_{|E|,1} & \cdots & \cdots & \boldsymbol{-Id_{|E|}} & 0_{|E|,1} \\
0_{1,|E|} & 0_{1,|E|} & \mathbf{1} & 0_{1,|E|} & \mathbf{1} & \cdots & \cdots & 0_{1,|E|} & \mathbf{1} \\
\vdots & \boldsymbol{A} & \boldsymbol{-b^{h_1}} & 0_{l,|E|} & \cdots & & & \cdots & 0_{l,1} \\
 & 0_{1,|E|} & \boldsymbol{-1} & 0_{1,|E|} & \cdots & & & & 0 \\
 & \vdots & 0_{l,1} & \boldsymbol{A} & \boldsymbol{-b^{h_2}} & 0_{l,|E|} & \cdots & \cdots & 0_{l,1} \\
 & & & 0_{1,|E|} & \boldsymbol{-1} & 0_{1,|E|} & \cdots & & 0 \\
 & & & \vdots & 0_{l,1} & \ddots & \ddots & & \vdots \\
 & & & & \vdots & & \ddots & \ddots & 0 \\
 & & & & & & & \boldsymbol{A} & \boldsymbol{-b^{h_{|Q^*|}}} \\
0_{1,|E|} & 0_{1,|E|} & 0 & 0_{1,|E|} & 0 & \cdots & \cdots & 0_{1,|E|} & \boldsymbol{-1}
\end{pmatrix}
\cdot
\begin{pmatrix}
x_1 \\ \vdots \\ x_{|E|} \\ y^{h_1} \\ y_0^{h_1} \\ y^{h_2} \\ y_0^{h_2} \\ \vdots \\ y^{h_{|Q^*|}} \\ y_0^{h_{|Q^*|}}
\end{pmatrix}
\begin{matrix}
= \\ = \\ = \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq
\end{matrix}
\begin{pmatrix}
\boldsymbol{|V|-1} \\ 0_{|E|} \\ \mathbf{1} \\ 0_{l,1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0
\end{pmatrix}
$$

Figure 5.2: Constraints matrix of $\mathcal{P}'$.

Large magnitude of the coefficients of the objective function and the budget constraint poses a challenge for the dynamic programming algorithm for the KP and the pseudo-

polynomial algorithm [50] for the M1BSTP. Surprisingly, our extended formulation shows that the difficulty of the M1BSTP lies in the number of distinct coefficient values in the budget contraint. Indeed, in the special case where the number k of distinct weights $w_1, \ldots, w_k$ in the budget constraint is a constant, the size of the set Q^* is bounded by $|V|^k$, so does the number of polyhedra in the disjunction. The number of columns of the extended formulation (5.18)-(5.22) is then polynomially bounded, and as the subtour separation problem can be solved in polynomial time, we get the following result:

Proposition 17. *If the number of distinct weights $w_1, \ldots, w_k$ in the budget constraint is a constant, then the M1BSTP can be solved in polynomial time by solving extended formulation (5.17)-(5.22).*

In the general case, even though formulation (5.18)-(5.22) is well structured, the exponential number of variables and constraints poses a challenge. Although the block diagonal structure of the matrix seems suitable for a decomposition method, the pricing problem related to any $P_h,\ h \in Q^*$ is NP-hard.

Proposition 18. *The pricing problem of $\{y^h \in \mathbb{R}^{|E|} \mid Ay^h \leq b^h\},\ h \in Q^*$ is at least as hard as solving a MBSTP.*

Proof. Proof The pricing problem is to find a vector $h = (h^1, \ldots, h^k) \in Q^*$ such that the reduced cost of a solution in the related polyhedron P_h is non-positive. Consider the following problem, where r_e denotes the reduced cost of variable $y^h_e,\ e \in E$ and P_{ST} the spanning tree polytope.

$$\min \sum_{e \in E} r_e y^h_e \tag{5.23}$$

$$s.t. \sum_{i=1}^{k} h^i = |V| - 1, \tag{5.24}$$

$$h^i \leq |E_i|, \qquad \forall i = 1, \ldots, k, \tag{5.25}$$

$$\sum_{i=1}^{k} w_i h^i \leq B, \tag{5.26}$$

$$\sum_{e \in E_i} y^h_e \leq h^i, \qquad \forall i = 1, \ldots, k, \tag{5.27}$$

$$y^h \in P_{ST}, \tag{5.28}$$

$$h^i \in \mathbb{N}, \tag{5.29}$$

$$y^h_e \in \{0, 1\}, \qquad \forall e \in E. \tag{5.30}$$

Constraints (5.24)-(5.26) correspond to conditions (5.1)-(5.3) and Constraints (5.27)-(5.28) ensure that the vector $h = (h^1, \ldots, h^k)$ belongs to Q^*. Therefore, the pricing problem corresponds to a spanning tree problem with multiple budget constraints, which at least as hard as MBSTP. □

Example 3

Consider the graph of Example 2. In what follows, we give the description of the three polyhedra in the disjunction. Several trivial or subtour inequalities redundant with constraints (5.5), written in bold, are removed.

$$P_{(2,1,0)} = \begin{cases} \boldsymbol{x_{AB} + x_{CD} \leq 2} \\ \boldsymbol{x_{AC} + x_{BC} + x_{BD} \leq 1} \\ \boldsymbol{x_{AD} \leq 0} \\ 0 \leq x_{AB} \\ x_{AB} \leq 1 \\ 0 \leq x_{AC} \\ 0 \leq x_{AD} \\ 0 \leq x_{BC} \\ 0 \leq x_{BD} \\ 0 \leq x_{CD} \\ x_{CD} \leq 1 \end{cases} \qquad P_{(2,0,1)} = \begin{cases} \boldsymbol{x_{AB} + x_{CD} \leq 2} \\ \boldsymbol{x_{AC} + x_{BC} + x_{BD} \leq 0} \\ \boldsymbol{x_{AD} \leq 1} \\ 0 \leq x_{AB} \\ x_{AB} \leq 1 \\ 0 \leq x_{AC} \\ 0 \leq x_{AD} \\ 0 \leq x_{BC} \\ 0 \leq x_{BD} \\ 0 \leq x_{CD} \\ x_{CD} \leq 1 \end{cases}$$

$$P_{(1,2,0)} = \begin{cases} \boldsymbol{x_{AB} + x_{CD} \leq 1} \\ \boldsymbol{x_{AC} + x_{BC} + x_{BD} \leq 2} \\ \boldsymbol{x_{AD} \leq 0} \\ x_{BD} + x_{BC} + x_{CD} \leq 2 \\ x_{AB} + x_{AC} + x_{BC} \leq 2 \\ 0 \leq x_{AB} \\ 0 \leq x_{AC} \\ x_{AC} \leq 1 \\ 0 \leq x_{AD} \\ 0 \leq x_{BC} \\ x_{BC} \leq 1 \\ 0 \leq x_{BD} \\ x_{BD} \leq 1 \\ 0 \leq x_{CD} \end{cases}$$

Then we detail all the inequalities of the related extended formulation (5.18)-(5.22).

$$
\begin{aligned}
\min\ & c_{AB}x_{AB} + c_{AC}x_{AC} + c_{AD}x_{AD} + c_{BD}x_{BD} + c_{BC}x_{BC} + c_{CD}x_{CD} \\
& x_{AB} + x_{AC} + x_{AD} + x_{BD} + x_{BC} + x_{CD} = 3, \\
& x_{AB} - y_{AB}^{(2,1,0)} - y_{AB}^{(2,0,1)} - y_{AB}^{(1,2,0)} = 0, \\
& x_{AC} - y_{AC}^{(2,1,0)} - y_{AC}^{(2,0,1)} - y_{AC}^{(1,2,0)} = 0, \\
& x_{AD} - y_{AD}^{(2,1,0)} - y_{AD}^{(2,0,1)} - y_{AD}^{(1,2,0)} = 0, \\
& x_{BD} - y_{BD}^{(2,1,0)} - y_{BD}^{(2,0,1)} - y_{BD}^{(1,2,0)} = 0, \\
& x_{BC} - y_{BC}^{(2,1,0)} - y_{BC}^{(2,0,1)} - y_{BC}^{(1,2,0)} = 0, \\
& x_{CD} - y_{CD}^{(2,1,0)} - y_{CD}^{(2,0,1)} - y_{CD}^{(1,2,0)} = 0, \\
& y_0^{(2,1,0)} + y_0^{(2,0,1)} + y_0^{(1,2,0)} = 1, \\
& y_{AB}^{(2,1,0)} + y_{CD}^{(2,1,0)} - 2y_0^{(2,1,0)} \leq 0, \\
& y_{AC}^{(2,1,0)} + y_{BD}^{(2,1,0)} + y_{BC}^{(2,1,0)} - y_0^{(2,1,0)} \leq 0, \\
& y_{AD}^{(2,1,0)} \leq 0, \\
& -y_e^{(2,1,0)} \leq 0, && \forall e \in \{AB, AC, CD, BD, BC, AD\}, \\
& y_e^{(2,1,0)} - y_0^{(2,1,0)} \leq 0, && \forall e \in \{AB, CD\}, \\
& y_0^{(2,1,0)} \geq 0, \\
& y_{AB}^{(2,0,1)} + y_{CD}^{(2,0,1)} - 2y_0^{(2,0,1)} \leq 0, \\
& y_{AC}^{(2,0,1)} + y_{BD}^{(2,0,1)} + y_{BC}^{(2,0,1)} \leq 0, \\
& y_{AD}^{(2,0,1)} - y_0^{(2,0,1)} \leq 0, \\
& -y_e^{(2,0,1)} \leq 0, && \forall e \in \{AB, AC, CD, BD, BC, AD\}, \\
& y_e^{(2,0,1)} - y_0^{(2,0,1)} \leq 0, && \forall e \in \{AB, CD\}, \\
& y_0^{(2,0,1)} \geq 0, \\
& y_{AB}^{(1,2,0)} + y_{CD}^{(1,2,0)} - y_0^{(1,2,0)} \leq 0, \\
& y_{AC}^{(1,2,0)} + y_{BD}^{(1,2,0)} + y_{BC}^{(1,2,0)} - 2y_0^{(1,2,0)} \leq 0, \\
& y_{AD}^{(1,2,0)} \leq 0, \\
& y_{BD}^{(1,2,0)} + y_{BC}^{(1,2,0)} + y_{CD}^{(1,2,0)} - 2y_0^{(1,2,0)} \leq 0, \\
& y_{AB}^{(1,2,0)} + y_{AC}^{(1,2,0)} + y_{BC}^{(1,2,0)} - 2y_0^{(1,2,0)} \leq 0, \\
& -y_e^{(1,2,0)} \leq 0, && \forall e \in \{AB, AC, CD, BD, BC, AD\}, \\
& y_e^{(1,2,0)} - y_0^{(1,2,0)} \leq 0, && \forall e \in \{AC, BD, BC\}, \\
& y_0^{(1,2,0)} \geq 0.
\end{aligned}
$$

5.2 Pseudo-polynomial size extended formulation for cactus graphs

The MdBSTP lies at the intersection of the MSTP and the multi-KP. In Section 3.3, we studied a special case of the dBSTP, adding more structure to the knapsack part of the problem. Now, considering a class of graphs with a particular topology, we devise a pseudo-polynomial algorithm for the MdBSTP. A graph $G = (V, E)$ is a *cactus graph* if any two cycles of G share at most one vertex. Figure 5.3 illustrates an example of a cactus graph.

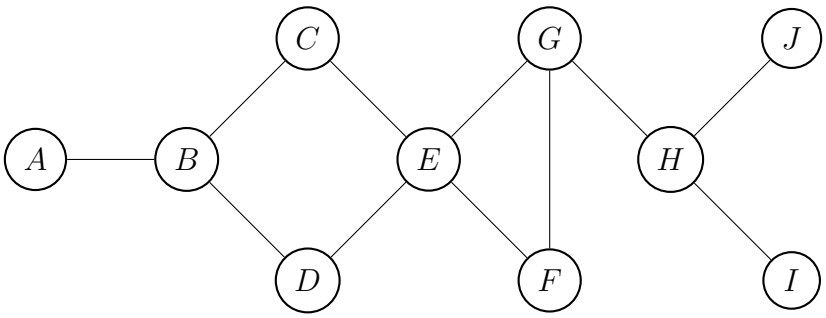


Figure 5.3: A cactus graph.

A cactus graph can be decomposed into independent blocks of bridge or blocks of cycle. By construction, every spanning tree solution contains all bridges and exactly $|C|-1$ edges from every cycle C in G. Exploiting this independent structure, the dBSTP can be solved using dynamic programming. To this end, we arbitrarily order the cycles $C_1, \ldots, C_l$ of G, and build an oriented graph $G_{ext} = (V_{ext}, A_{ext})$ exploring every cycle of G one by one. V_{ext} contains a sink node s, a target node t and several inner nodes $(j, b_m^1, \ldots, b_m^d)$ with $j = 1, \ldots, l$ and $b_m^i \in \mathbb{N}$, $b_m^i \leq B'^i = B^i - \sum_{e \in E | e \text{ is a bridge}} w_e^i,\ i = 1, \ldots, d$.

We add an arc from the vertex $(j, b_m^1, \ldots, b_m^d)$ to the vertex $(j+1, b_p^1, \ldots, b_p^d)$ if there exists $S \subset C_{j+1}, |S| = |C_{j+1}| - 1$ such that $b_p^i = b_m^i + \sum_{e \in S} w_e^i$ for every $i = 1, \ldots, d$. We also add arcs from the source s to every node related to the cycle C_1, $(1, b_m^1, \ldots, b_m^d)$ where $b_m^i = \sum_{e \in S_1} w_e^i,\ i = 1, \ldots, d$, $S_1 \subset C_1,\ |S_1| = |C_1| - 1$, and arcs from every node associated with C_l to the sink node t. The construction of G_{ext} is illustrated in Figure 5.4:

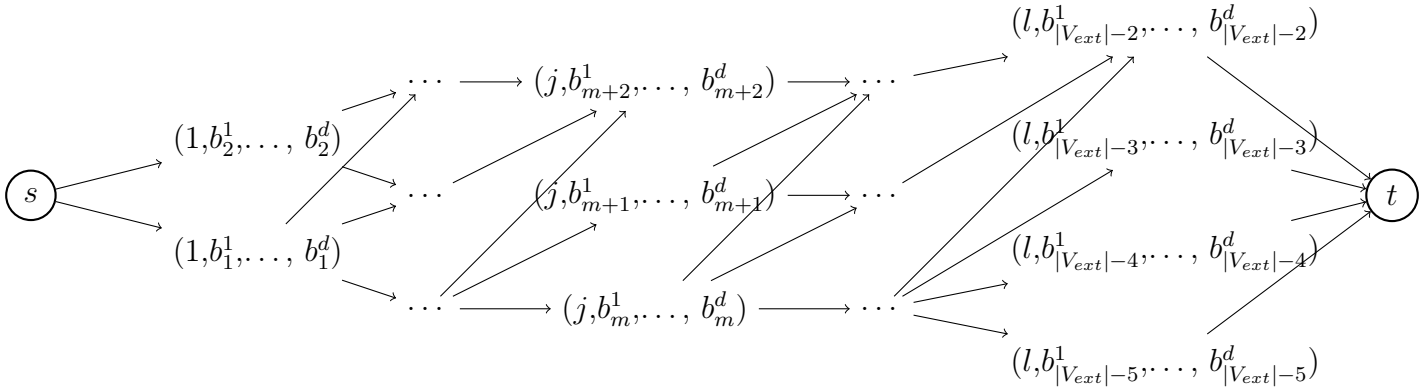


Figure 5.4: Example of an extended graph $G_{ext} = (V_{ext}, A_{ext})$. The figure illustrates the structure of an extended graph $G_{ext} = (V_{ext}, A_{ext})$, representing all subproblems and transitions of the dynamic programming approach for solving dBSTP when G is a cactus graph.

The following propositions are implied by the construction of G_{ext}:

Proposition 19. *Every feasible solution $T = (V, F)$ of the dBSTP can be associated with a directed $s-t-$path.*

Proof. Let $S_j = F \cap C_j,\ j = 1,\ldots,l$, by hypothesis those sets satisfy $|S_j| = |C_j| - 1$ and $\sum_{e\in\bigcup_{j=1}^{l} S_j} w_e^i \leq B'^i$, for $i = 1,\ldots,d$. By construction of G_{ext} there is a $s-t-$path such that every j-th arc is associated with $S_j,\ j = 1,\ldots,l$. □

Proposition 20. *Every directed $s-t-$path of G_{ext} can be associated with at least one feasible solution for the dBSTP.*

Proof. G_{ext} is a directed acyclic graph and every directed $s-t-$path contains exactly $l+1$ arcs. Any j-th arc, $j = 1,\ldots,l$, of a $s-t-$path is associated with a subset $S_j \subset C_j, |S_j| = |C_j| - 1$. Edges of S_j cover all vertices of the cycle C_j. Considering all the arcs of any $s-t-$path, we obtain a forest $F = (V, \cup_{j=1,\ldots,l} S_j)$ that covers the vertices of every cycle of G. Moreover, $\sum_{e\in\bigcup_{j=1}^{l} S_j} w_e^i \leq B'^i$ for $i = 1,\ldots,d$, by adding every bridge of G to F, we obtain a spanning tree satisfying the d budget constraints. □

Notice that a directed $s-t-$path may be associated with several feasible solutions as several subsets $S_j, S'_j \subset C_j,\ j \in \{1,\ldots,l\}$ satisfying $|S_j| = |S'_j| = |C_j| - 1$ may verify $\sum_{e\in S_j} w_e^i = \sum_{e\in S'_j} w_e^i,\ i = 1,\ldots,d$.

The MdBSTP can be solved by computing a minimum cost $s-t-$path in G_{ext}. In this sense, we propose a flow-based linear program. Additional variables $y_a \in \mathbb{R}_+$ are introduced for each arc $a \in A_{ext}$. In order to link the variables y with the original variables x, for every cycle C of G and every weight $w \in \{w_e \in \mathbb{N}^d \mid e \in C\}$, we define $E_w^C = \{e \in C \mid w_e = w\}$ and $A_w^C = \{(u,v) \in A_{ext} \mid u = (j-1, b^1,\ldots,b^d), v = (j, b^1 - w^1 + \sum_{e\in C} w_e^1, \ldots, b^d - w^d + \sum_{e\in C} w_e^d), j \text{ is associated with } C\}$. Finally, let $F = \{e \in E \mid e \text{ is a bridge}\}$ be the set of bridges in G.

$$\min \sum_{e\in E} c_e x_e \tag{5.31}$$

$$x(E_w^C) = |E_w^C| - y(A_w^C), \qquad \forall \text{ cycle } C, \forall w \in \{w_e \in \mathbb{N}^d \mid e \in C\}, \tag{5.32}$$

$$y(\delta^-(t)) = 1, \tag{5.33}$$

$$y(\delta^-(v)) = y(\delta^+(v)), \qquad \forall v \in V_{ext}\backslash\{s,t\}, \tag{5.34}$$

$$x_e = 1, \qquad \forall e \in F, \tag{5.35}$$

$$0 \leq y_a, \qquad \forall a \in A_{ext}, \tag{5.36}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E\backslash F. \tag{5.37}$$

Constraints (5.32) link the design variables x to the variables y of the flow part. Indeed, given an integer solution (x^*, y^*) satisfying (5.32)-(5.37), constraints (5.33), (5.34) and (5.36) ensure that the set $\{y_a \in A_{ext} \mid y_a^* > 0\}$ corresponds to a directed $s-t-$path. Every j-th arc of the latter $s-t-$path implies that a set $S_j \subset C_j,\ |S_j| = |C_j| - 1$ belongs to the spanning tree solution related to x^*. Constraints (5.32) ensure that $x_e^* = 1,\ e \in S_j$ and $x^*_{C_j\backslash S_j} = 0$.

Proposition 21. *The system (5.32)-(5.37) is integer.*

Proof. Assume that the system (5.32)-(5.37) has a fractional extreme point (x^*, y^*). First, assume that y^* is fractional, in this case, constraints (5.33) and (5.34) ensure that there are several directed $s-t-$paths $p_1, \ldots, p_q$ in G_{ext} such that all variables associated with the arcs of $p_i,\ i = 1, \ldots, q$ have the same positive value. Consider two directed paths p_r and p_s with $r, s \in \{1, \ldots, q\}, r \neq s$. There is at least one cycle $C_j,\ j = 1, \ldots, l$ such that $\sum_{e\in S_j^r} w_e \neq \sum_{e\in S_j^s} w_e$ where S_j^r (respectively S_j^s) is the set related to the j-th arc of the path p_r (respectively p_s). Constraints (5.32) ensure that there are at least two edges $e \in E_{w_e}^{C_j}$ and $f \in E_{w_f}^{C_j}$, where $w_e = \sum_{g\in C_j} w_g - \sum_{g\in S_j^r} w_g$ and $w_f = \sum_{g\in C_j} w_g - \sum_{g\in S_j^s} w_g$, with fractional value.

Now, consider the solutions (x^1, y^1) and (x^2, y^2) such that:

- $y_a^1 = y_a^* + \epsilon, \forall a \in p_r\backslash p_s$, $y_a^1 = y_a^* - \epsilon, \forall a \in p_s\backslash p_r$ and $y_a^1 = y_a^*$ otherwise,
- $x_e^1 = x_e^* - \epsilon$ and $x_f^1 = x_f^* + \epsilon$ for an edge $e \in E^{C_j}_{\sum_{g\in C_j} w_g - \sum_{g\in S_j^r} w_g}$ and an edge $f \in E^{C_j}_{\sum_{g\in C_j} w_g - \sum_{g\in S_j^s} w_g}$ where $\sum_{g\in S_j^r} w_g \neq \sum_{g\in S_j^s} w_g,\ \forall j = 1, \ldots, l$, and $x_g^1 = x_g^*$ otherwise,
- $y_a^2 = y_a^* - \epsilon, \forall a \in p_r\backslash p_s$, $y_a^2 = y_a^* + \epsilon, \forall a \in p_s\backslash p_r$ and $y_a^2 = y_a^*$ otherwise,
- $x_e^2 = x_e^* + \epsilon$ and $x_f^2 = x_f^* - \epsilon$ for an edge $e \in E^{C_j}_{\sum_{g\in C_j} w_g - \sum_{g\in S_j^r} w_g}$ and an edge $f \in E^{C_j}_{\sum_{g\in C_j} w_g - \sum_{g\in S_j^s} w_g}$ where $\sum_{g\in S_j^r} w_g \neq \sum_{g\in S_j^s} w_g,\ \forall j = 1, \ldots, l$, and $x_g^1 = x_g^*$ otherwise.

(x^1, y^1) and (x^2, y^2) are feasible solutions of (5.32)-(5.37). Furthermore, $(x^*, y^*) = \frac{1}{2}(x^1, y^1) + \frac{1}{2}(x^2, y^2)$. Hence (x^*, y^*) is not an extreme point.

Now, assume that y^* is integer and x^* is fractional. The integrality of y^* induces a unique directed $s-t-$path with strictly positive value in G_{ext}. Therefore every cycle $C_l,\ l=1,\ldots,l$ is associated with a unique set $S_j,\ |S_j|=|C_j|-1$. Constraints (5.32) imply that $x_e=1$ for every edge $e\in\{f\in C_j \mid w_f \neq \sum_{g\in C_j} w_g - \sum_{g\in S_j} w_g\}$. We deduce that only edges that belong to $\{f\in C_j \mid w_f = \sum_{g\in C_j} w_g - \sum_{g\in S_j} w_g\}$ may have fractional values. Consider two fractional variables $x^*_{e_1}$ and $x^*_{e_2} \in \{f\in C_j \mid w_f = \sum_{g\in C_j} w_g - \sum_{g\in S_j} w_g\}$ (notice that if $|\{f\in C_j \mid w_f = \sum_{g\in C_j} w_g - \sum_{g\in S_j} w_g\}|<2$ there is no fractional variables, contradicting our assumption). Let (x^1,y^1) and (x^2,y^2) be feasible solutions of (5.32)-(5.37) such that:

- $x^1_{e_1}=x^*_{e_1}+\epsilon$, $x^1_{e_2}=x^*_{e_2}-\epsilon$, $x^1_g=x^*_g$ otherwise,
- $y^1=y^*$,
- $x^2_{e_1}=x^*_{e_1}-\epsilon$, $x^2_{e_2}=x^*_{e_2}+\epsilon$, $x^2_g=x^*_g$ otherwise,
- $y^2=y^*$.

We have that $(x^*,y^*)=\frac{1}{2}(x^1,y^1)+\frac{1}{2}(x^2,y^2)$. Since $(x^1,y^1)\neq(x^2,y^2)$, this contradicts the fact that (x^*,y^*) is an extreme point. □

The number of cycles in a cactus graph is bounded by $\lfloor\frac{|E|}{3}\rfloor$. Hence, the number of nodes in V_{ext} is bounded by a pseudo-polynomial value in the order of $O(|E|\prod_{i=1}^{d} B'^i)$. By construction, every vertex of G_{ext} may be adjacent to up to $|E|$ outgoing arcs, then the size of $|A_{ext}|$ is bounded by $O(|E|^2\prod_{i=1}^{d} B'^i)$. Moreover, the number of constraints (5.32) is bounded by $|E|\backslash|F|$. Hence, an implementation of the linear program (5.31)-(5.37) solves the MdBSTP on cactus graph in pseudo-polynomial time.

Example 4

Consider the 1BSTP instance given by Figure 5.5 with weights on the edges and a total budget of 25.

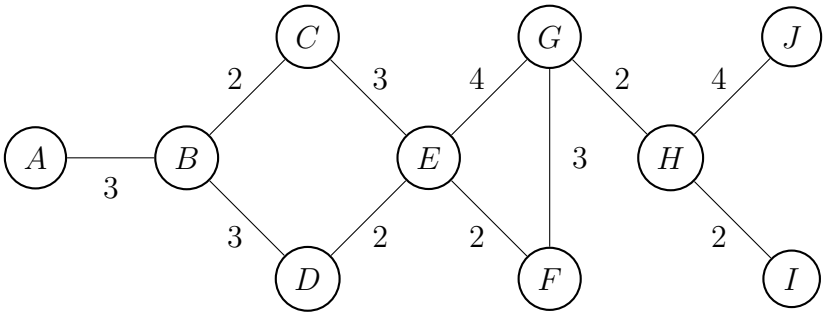


Figure 5.5: An instance of the 1BSTP with a cactus topology.

As a preprocessing, we remove all bridges AB, GH, HI and HJ and update the budget $B'=25-11=14$. Figure 5.6 represents the extended graph $G_{ext}=(V_{ext},A_{ext})$ related to instance given by Figure 5.5.

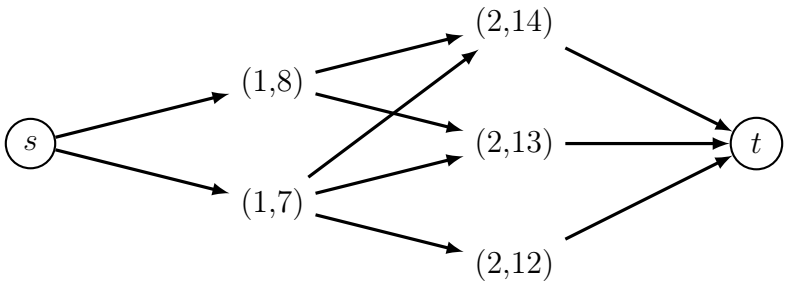


Figure 5.6: Extended graph $G_{ext} = (V_{ext}, A_{ext})$ related to instance 5.5.

Vertices $(1, 8)$ and $(1, 7)$ are related to the cycle $BCDE$. As $w_{BC} = w_{DE}$ and $w_{BD} = w_{CE}$, every spanning tree consumes either 7 or 8 units of resource in order to cover the cycle. Vertices $(2, 12)$, $(2, 13)$ and $(2, 14)$ are associated with the cycle EFG. In this case, every spanning tree consumes either 5, 6 or 7 units of resource. Notice that a solution requiring 8 units of resource for the first cycle and 7 for the second is not feasible, the resulting vertex $(2, 15)$ does not belong to V_{ext}.

The associated linear program is given by:

$$\min \sum_{e \in E} c_e x_e \tag{5.38}$$

$$x_{BC} + x_{DE} = 2 - y_{s,(1,8)}, \tag{5.39}$$

$$x_{BD} + x_{CE} = 2 - y_{s,(1,7)}, \tag{5.40}$$

$$x_{EG} = 1 - y_{(1,8),(2,13)} - y_{(1,7),(2,12)}, \tag{5.41}$$

$$x_{GF} = 1 - y_{(1,8),(2,14)} - y_{(1,7),(2,13)}, \tag{5.42}$$

$$x_{EF} = 1 - y_{(1,7),(2,14)}, \tag{5.43}$$

$$y_{(2,14),t} + y_{(2,13),t} + y_{(2,12),t} = 1, \tag{5.44}$$

$$y_{s,(1,8)} = y_{(1,8),(2,14)} + y_{(1,8),(2,13)}, \tag{5.45}$$

$$y_{s,(1,7)} = y_{(1,7),(2,14)} + y_{(1,7),(2,13)} + y_{(1,7),(2,12)}, \tag{5.46}$$

$$y_{(1,8),(2,14)} + y_{(1,7),(2,14)} = y_{(2,14),t}, \tag{5.47}$$

$$y_{(1,8),(2,13)} + y_{(1,7),(2,13)} = y_{(2,13),t}, \tag{5.48}$$

$$y_{(1,7),(2,12)} = y_{(2,12),t}, \tag{5.49}$$

$$0 \leq y_a \qquad \forall a \in A_{ext}, \tag{5.50}$$

$$x_{AB} = 1, \tag{5.51}$$

$$x_{GH} = 1, \tag{5.52}$$

$$x_{HI} = 1, \tag{5.53}$$

$$x_{HJ} = 1, \tag{5.54}$$

$$0 \leq x_e \leq 1 \qquad \forall e \in \{BC, CE, BD, DE, EG, EF, GF\}. \tag{5.55}$$

Chapter 6

Valid inequalities for the 1-budgeted spanning tree problem

A key ingredient of modern algorithms to solve combinatorial optimization problems is the use of strong valid inequalities, also known as cutting planes, to strengthen the continuous relaxation of the problem. The convex hull $\mathcal{H}$ of the feasible solutions of the 1BSTP lies at the intersection of the spanning tree and knapsack polyhedra. Any valid inequality for the SPT or the KP is also a valid inequality for the 1BSTP. The convex hull of the spanning trees is completely characterized. However, the KP is NP-hard and one strand of the literature on cutting planes deals with its polytopes [44]. Generating cuts within a Branch-and-Cut procedure is a delicate task. The challenge is to balance two conflicting criteria: the separation running time and the benefits of the generated cuts. It is not tractable to generate all the subtour inequalities to solve the problem. Even though the separation problem of these inequalities is polynomially solvable [59], the separation procedure is expensive for large graphs. Therefore, we devise fast and efficient heuristics for separating the spanning tree's valid inequalities. Cover inequalities [44] are well known valid inequalities for the knapsack polytope. Unfortunately, their separation problems are NP-hard. Furthermore, experimental results show that only a few of these inequalities are violated by the fractional solutions encountered during the course of our cutting plane procedure. In order to strengthen these inequalities, we consider the spanning tree polytope in the definition of the cover inequalities.

In order to efficiently solve our problem, the major challenge is to efficiently generate valid inequalities that exploit both structural properties of the knapsack and the spanning tree polytopes. The convex hull $\mathcal{H}$ of the 1BSTP is completely defined by the projection of an exact but exponentially large formulation (see Proposition 16). In practice, computing this projection is time consuming and only a small fraction of the resulting inequalities define facets of the convex hull of the feasible solutions. The double description method [55], is a popular generic algorithm to enumerate the extreme rays of a polyhedron. Unfortunately, its implementation requires as input all the data of the exponentially large extended formulation. By carefully selecting rays of the projection cone, we generate valid inequalities that are added a priori to strengthen the linear formulation of the problem.

Due to exponential size of the extended formulation, it is prohibitive to generate these inequalities to cut off fractional solutions encountered during the course of the algorithm. An interesting open question is how to exploit the exact formulation in order to efficiently

generate valid inequalities. In order to circumvent this difficulty, we propose a new extended formulation based on a tractable number of polyhedra, that allows one to cut off any fractional solution. However the price to pay is that the resulting valid inequality may not be deep.

6.1 Separating efficiently spanning tree's inequalities

When the subtour formulation is solved by a Branch-and-Cut algorithm, violated inequalities must be identified dynamically in the course of the branching algorithm. The separation of the most violated inequalities is usually performed by maximum flow algorithms and requires a prohibitive $O(|V|^4)$ time [59]. We propose instead heuristic routines for finding efficiently violated inequalities of the STP by a solution x^*. In addition to subtour inequalities, we consider also separating multicut and cut inequalities. We exploit the structure of x^* and its support graph $G^* = (V, E^*)$ where $E^* = \{e \in E \mid x^*_e > 0\}$. In the first nodes of the search tree, some computed solutions x^* are integer but their support graph is not connected. Therefore, at least one connected component contains a cycle.

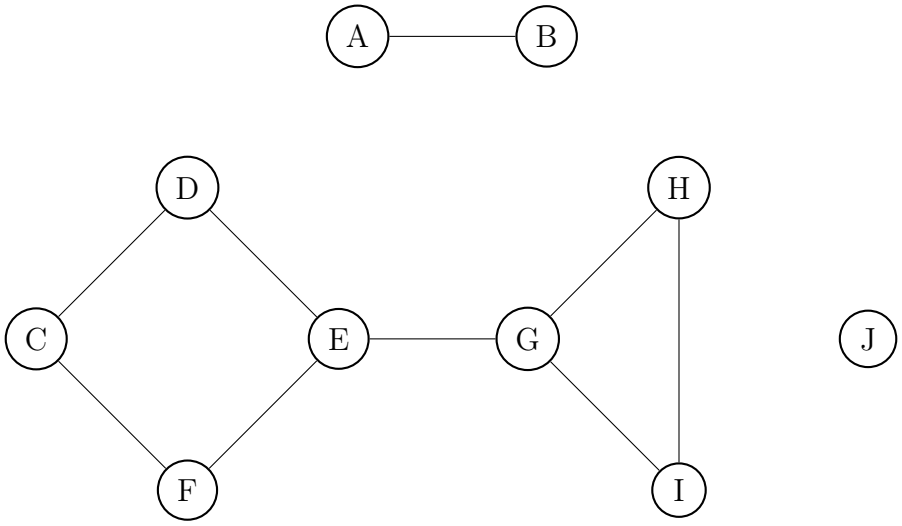


Figure 6.1: Example of a support graph associated with an infeasible integer solution x^*.

Figure 6.1 illustrates the support graph of an integer solution x^* that is not a spanning tree. The subtour inequality $x_{CD} + x_{DE} + x_{EF} + x_{CF} + x_{EG} + x_{GH} + x_{HI} + x_{GI} \leq 6$ cuts x^*.

In practice, adding a subtour constraint associated with a given connected component performs better than adding multiple subtour constraints associated with the cycles inside it. Algorithm 11 implements the separation heuristic in $O(|V||E|)$ time when x^* is integer. The resulting inequality in G^* is lifted by considering all the edges e in the original graph that belong to the connected component but with $x^*_e = 0$.

Algorithm 11: Separation algorithm for an integer solution x^*

Data: E, $x^* \in \{0,1\}^{|E|}$, $G^* = (V, E^*)$ where $E^* = \{e \in E \mid x^*(e) > 0\}$

```
if there are p > 1 connected components in G* then
    for every connected component G*[V_i], i = 1, ..., p do
        if component G*[V_i] contains more than |V_i| − 1 edges then
            Add the subtour constraint Σ_{uv∈E|u,v∈V_i} x_uv ≤ |V_i| − 1;
        end
    end
end
```

After adding a number of valid inequalities, the computed solution x^* becomes in general fractional and its support graph G^* contains 2 or more connected components $G^*[V_1], \dots G^*[V_p]$. The following result shows that a subtour constraint is violated in at least one connected component.

Proposition 22. *If $p \geq 2$, at least one connected component $G^*[V_i]$, $i = 1, \dots, p$ violates the related subtour inequality $\sum_{uv \in E^* | u,v \in V_i} x_{uv} \leq |V_i| - 1$.*

Proof. Assume by contradiction that every connected component satisfies the related subtour inequality $\sum_{uv \in E^* | u,v \in V_i} x_{uv} \leq |V_i| - 1$. By summing all those inequalities we obtain $\sum_{e \in E^*} x_e = \sum_{i=1}^{p} \sum_{uv \in E^* | u,v \in V_i} x_{uv} \leq |V| - p$. We have a contradiction as $\sum_{e \in E^*} x_e = |V| - 1$ and $p \geq 2$. □

The multicut inequality $\sum_{uv \in E | u \in V_i, v \in V_j, i \neq j} x_{uv} \geq p - 1$ is redundant with all the subtour inequalities $\sum_{uv \in E | u,v \in V_i} x_{uv} \leq |V_i| - 1$, $i = 1, \dots, p$ and $\sum_{e \in E} x_e = |V| - 1$. However, x^* may satisfy some of these inequalities and thus, the solver may not add them. In order to separate x^* and strengthen the formulation of the problem, it is relevant to add only the violated subtour inequalities $\sum_{uv \in E | u,v \in V_i} x_{uv} \leq |V_i| - 1$ and the multicut inequality. In this case, the latter inequality is relevant as illustrated in the following example.

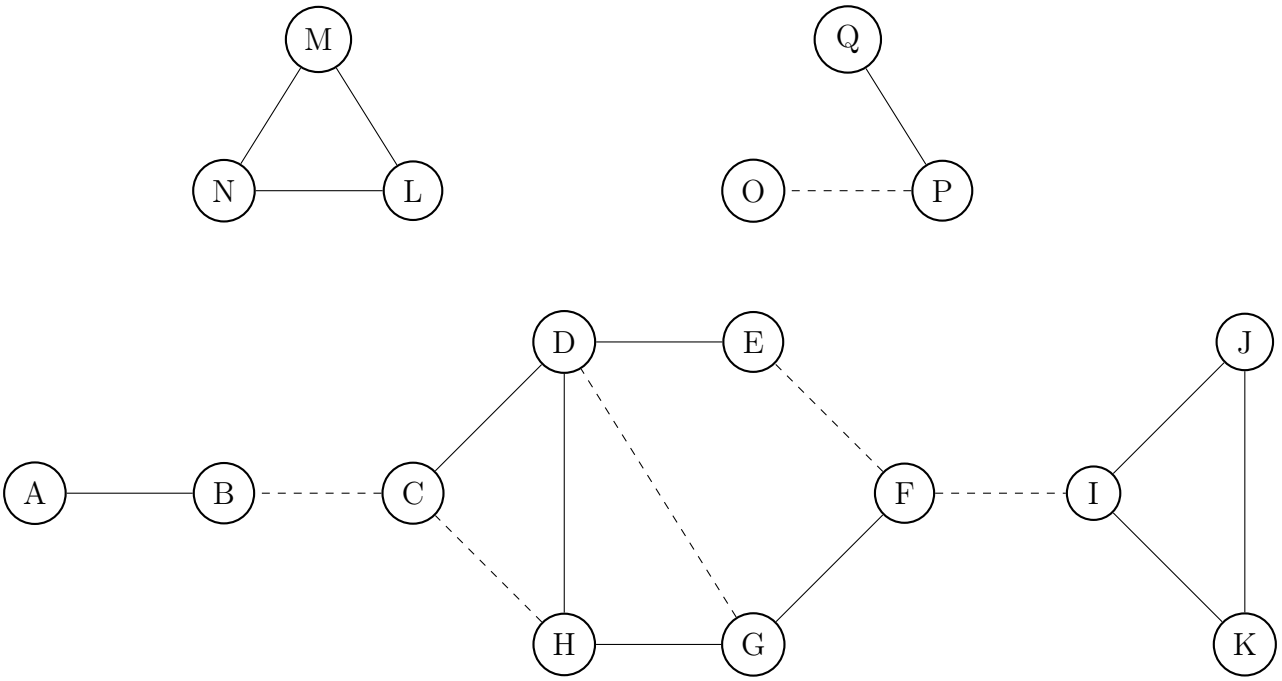


Figure 6.2: Example of a disconnected support graph associated with an infeasible fractional solution x^*. The branches of the support graph are formed by edges OP, PQ, AB and BC.

Consider a fractional solution x^* and its support graph given by Figure 6.2 where the dashed edges correspond to fractional variables $x^*_e = \frac{1}{2}$ and the bold edges correspond to integer variables $x^*_e = 1$. The solution x^* violates the subtour constraints $x_{LM} + x_{LN} + x_{MN} \leq 2$ and $x_{AB} + x_{BC} + x_{CD} + x_{DE} + x_{EF} + x_{GF} + x_{GH} + x_{CH} + x_{DH} + x_{DG} + x_{FI} + x_{IJ} + x_{JK} + x_{IK} \leq 10$ associated with the two connected components. Note that the multicut inequality $\sum_{e \in \bigcup_{i=1}^{3} \delta^G(V_i)} x_e \geq 2$, where $\delta^G(S) = \{uv \in E \mid u \in S, v \notin S\}$, is not implied by the above subtour constraints.

Bridges BC, FI and OP induce violated cut inequalities. In contrast with the subtour and multicut inequalities, the cut inequalities do not completely characterize the spanning tree polytope. However, they can be detected in linear time by considering only the branches of G^*, see Algorithm 3. In this case, we only consider the cuts induced by every connected component after removing fractional edges that belong to a branch, BC and OP in Figure 6.2. Hence we generate the inequalities $\sum_{e \in \delta^G(\{L,M,N\})} x_e \geq 1$, $\sum_{e \in \delta^G(\{Q,P\})} x_e \geq 1$, $\sum_{e \in \delta^G(\{O\})} x_e \geq 1$, $\sum_{e \in \delta^G(\{A,B\})} x_e \geq 1$, $\sum_{e \in \delta^G(\{C,D,E,F,G,H,I,J,K\})} x_e \geq 1$.

Algorithm 12 summarizes the separation of basic spanning tree inequalities in this case. In practice, it is very effective and requires $O(|V|^2|E|)$ time. The resulting inequalities are also lifted by considering all the edges in E.

Algorithm 12: Separation algorithm for a fractional solution when G^* is not connected

Data: E, $x^* \in [0:1]^{|E|}, G^* = (V, E^*)$ where $E^* = \{e \in E | x^*(e) > 0\}$

for every connected component $G^*[V_i]$, $i = 1, \ldots, p$ **do**
  if $\sum_{uv \in E^* | u,v \in V_i} x_{uv} > |V_i| - 1$ **then**
    Add the subtour constraint $\sum_{uv \in E | u,v \in V_i} x_{uv} \leq |V_i| - 1$;
  end
  Let $S = \{e \in E^*[V_i] \mid e$ belongs to a branch and $x^*_e < 1\}$;
  Consider the subgraph $G' = (V_i, E^*[V_i] \backslash S)$;
  for every connected component $G'[V_i^j]$, $j = 1, \ldots, l$ of G' **do**
    Add the cut constraint $\sum_{uv \in E | u \in V_i^j, v \notin V_i^j} x_{uv} \geq 1$;
  end
end
Add the multicut constraint $\sum_{uv \in E | u \in V_i, v \in V_j, i \neq j} x_{uv} \geq p - 1$;

After few calls of Algorithm 12, the support graph G^* associated with the computed fractional solution x^* becomes connected. In this case, we propose in Algorithm 13 an heuristic that separates, under some conditions, subtour and multicut inequalities. The algorithm is very efficient in practice and requires $O(|E|^2)$ time in the worst case.

Algorithm 13: Heuristic - Separation algorithm for fractional solution when G^* is connected

Data: E, $x^* \in [0:1]^{|E|}$, $G^* = (V, E^*)$ where $E^* = \{e \in E \mid x^*(e) > 0\}$, G^* connected

Compute $T^* = (V, F^*)$ the spanning tree of G^* maximizing the value $\sum_{e \in F^*} x^*_e$;
for $e \in E^* \backslash F^*$ **do**
  if the created cycle C obtained by adding e to T^* violates a subtour constraint **then**
    Add the associated subtour constraint $\sum_{uv \in E |\ u,v \in V[C]} x_{uv} \leq |C| - 1$ to the formulation;
  end
end
Let $S = \{e \in E^* \mid e$ belongs to a branch and $x^*_e < 1\}$;
if $|S| > 0$ **then**
  Consider the subgraph $G' = (V, E^* \backslash S)$ with $|S| + 1$ connected components $G'[V_i]$, $i = 1, \ldots, |S| + 1$;
  Add the multicut constraint $\sum_{\{u,v\} \in E | u \in V_i, v \in V_j, i \neq j} x_{uv} \geq |S|$;
  Let $S' = \{e \in E^* \mid e$ belongs to a branch$\}$;
  Let $V' = \{v \in V \mid v$ is not an isolated vertex in the subgraph $G'' = (V, E^* \backslash S')\}$;
  Add the subtour constraint $\sum_{uv \in E | u,v \in V'} x_{uv} \leq |V| - |S'| - 1$;
end

First, Algorithm 13 computes a spanning tree $T^* = (V, F^*)$ maximizing $\sum_{e \in F^*} x^*_e$ to

identify violated subtour constraints. The latter are generated by exploring every cycle obtained by adding an edge in $E^* \backslash F^*$ to T^*. Then, Algorithm 13 removes every edge with a fractional value that belongs to a branch of G^*, if any, and generates a multicut constraint with the created components. Finally, Algorithm 13 removes all edges that belong to a branch of G^* and adds a subtour inequality related to the remaining connected component.

The following example illustrates the implementation of Algorithm 13.

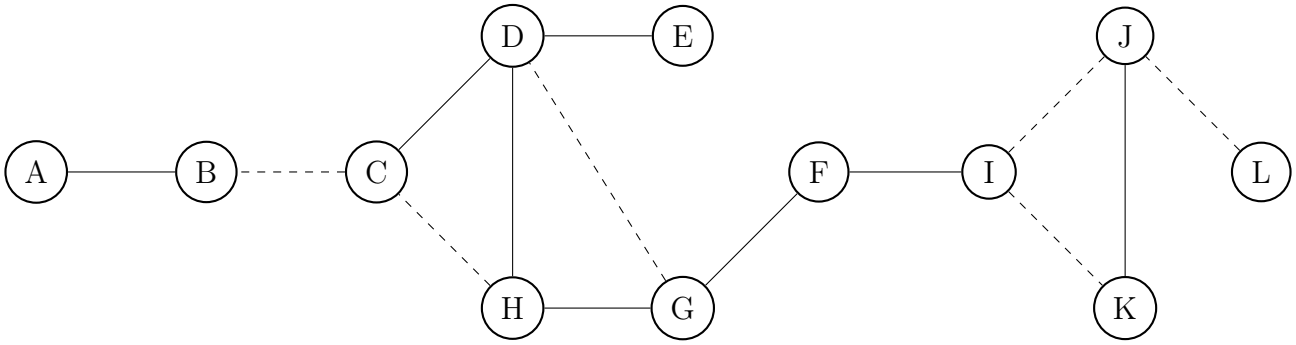


Figure 6.3: Example of a connected support graph associated with a infeasible fractional solution x^*. The branches of the support graph are formed by edges AB, BC, DE and JL. A dashed edge e corresponds to a fractional variable with value $x_e^* = \frac{1}{2}$ and a bold edge e corresponds to an integer variable with value $x_e^* = 1$.

In Figure 6.3, an example of T^* is given by $F^* = \{AB, BC, CD, DE, DH, GH, FG, FI, IJ, JK, JL\}$. By exploring the cycle in the neighborhood of T^*, Algorithm 13 generates the subtour inequalities $x_{CD} + x_{DH} + x_{CH} \leq 2$ and $x_{DG} + x_{DH} + x_{GH} \leq 2$.

Then, it removes the edges BC and JL and adds the violated multicut inequality $\sum_{e \in \delta^G(\{A,B\}) \cup \delta^G(\{L\}) \cup \delta^G(\{C,D,E,F,G,H,I,J,K\})} x_e \geq 2$ to the model. After removing the edges AB, BC, DE and JL that belong to branches, Algorithm 13 generates the violated subtour inequality $\sum_{uv \in E | u,v \in \{C,D,F,G,H,I,J,K\}} x_{uv} \leq 7$.

6.2 Separating efficiently cover inequalities

The KP is a fundamental problem in discrete optimization and appears as a subproblem in general binary programs $\max\{cx \mid Ax \leq b,\ x \in \{0,1\}^n\},\ n \in \mathbb{N}$. In our context, it corresponds to complicating constraints added to the STP. There are many papers dealing with valid inequalities for the knapsack polytope [44]. Most of these focus on cover inequalities.

Definition 2. [44] A set $S \subseteq E$ is a cover if $\sum_{e \in S} w_e > B$. A cover is minimal if $S \backslash \{e\}$ is not a cover for any $e \in S$.

In general, the cover inequalities $\sum_{e \in S} x_e \leq |S| - 1$ can be enhanced by lifting procedure [34]. Clearly, the cover inequalities are valid for our problem. However, they focus only on the knapsack part. The following class of cover inequalities encompasses also the spanning tree side.

Proposition 23. *[2] Given a forest $S \subseteq E$ such that every extension of S is a spanning tree violating the budget constraint. The inequality $\sum_{e \in S} x_e \leq |S| - 1$ is also valid for the M1BSTP.*

In practice, few cover inequalities obtained by Proposition 23 cut fractional solution in a Branch-and-Cut algorithm, especially if $|S|$ is small. Hence, we only separate the case where S is the edge set of a spanning tree. Given a fractional solution x^*, an interesting practical way to implement cover inequality separation is to compute $\max\{\sum_{e \in S} x^*_e \mid T = (V, S)$ is a spanning tree in $G^*\}$. If $\sum_{e \in S} x^*_e > |V| - 2$ and $\sum_{e \in S} w_e > B$ then the related cover inequality cuts x^*. This argument could be generalized to any infeasible spanning tree $T = (V, S)$ such that $\sum_{e \in S} x^*_e > |V| - 2$ and $\sum_{e \in S} w_e > B$. The resulting inequality can be lifted using Algorithm 14 in $O(|E|^2)$.

Algorithm 14: Lifting procedure for a valid inequality $\sum_{e \in S} x_e \leq |V| - 2$

Data: $G = (V, E), w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}, G' = (V, S)$ a connected graph, $\sum_{e \in S} x_e \leq |V| - 2$ a valid inequality

```
Compute a spanning tree T^w = (V, F^w) minimizing Σ_{e∈F^w} w_e on G' = (V, S);
for e ∈ E\S do
    Let f ∈ argmax{w_g | g ∈ C\{e}} where C is the unique cycle created by
      adding e to T^w;
    if Σ_{h∈F^w∪{e}\{f}} w_h > B then
        S ← S ∪ {e};
        if w_e < w_f then
            F^w ← F^w ∪ {e}\{f};
        end
    end
end
```

Algorithm 14 starts with the edge set S and examines the edges in $E \backslash S$ in a given order. It iteratively tries to add edges to S such that at each step $\min\{\sum_{e \in F} w_e \mid T = (V, F)$ is a spanning tree in $G' = (V, S)\} > B$. If the algorithm succeeds to augment, then the initial cover inequality is strengthened. Changing the order of examining the edges may yield a different valid inequality.

The next result illustrates an interesting particular class of edge sets S.

Proposition 24. *Given $S \subseteq E$ such that the graph $G' = (V, S)$ is connected. The cover inequality $\sum_{e \in S} x_e \leq |V| - 2$ is valid if and only if there is no feasible solution of the 1BSTP in the subgraph $G' = (V, S)$.*

Given a fractional solution x^* such that its support graph $G^* = (V, E^*)$, where $E^* = \{e \in E \mid x^*_e > 0\}$, is connected, one can verify the condition of Proposition 24 by computing a minimum weight spanning tree on G'.

A different idea to implement Proposition 24 is to remove carefully a set of edges $R \subset E^*$ such that $\min\{\sum_{e \in F} w_e \mid T = (V, F)$ is a spanning tree in $E^* \backslash R\} > B$ and $\sum_{e \in R} x^*_e < 1$. In this case, Proposition 24 could be applied to the set $S = E^* \backslash R$. In order to compute the desired set R, it is sufficient to solve the *spanning tree interdiction problem*. The goal is to

find a subset of edges $R \subset E^*$ such that the weight of any spanning tree in $G'' = (V, E^* \backslash R)$ is higher than B. That is

$$\min_{R \subset E^*} \{ \sum_{e \in R} x_e^* \mid \sum_{e \in F} w_e > B, \forall\ T = (V, F) \text{ spanning tree in } G'' = (V, E^* \backslash R) \}. \tag{6.1}$$

If $\sum_{e \in R} x_e^* < 1$, then Proposition 24 could be applied to the set $S = E^* \backslash R$. Otherwise, x^* could not be cut by using Proposition 24. The spanning tree interdiction problem is strongly NP-hard [27] and prohibitive to solve routinely. Instead, we consider an efficient heuristic. Algorithm 15 builds iteratively a set $R \subset E^*$ of edges belonging to the current minimum weight spanning tree in G^*. These edges are greedily selected at each step in such a way that their removal generates a large increase of the value of the minimum spanning tree value in the graph $G'' = (V, E^* \backslash R)$. Let $T^w = (V, F^w)$ denote a minimum spanning tree for the edge weights w_e in the support graph G^*. For any edge $e \in F^w$, let δ_e denotes the cut in G^* induced by the two components in $F^w \backslash \{e\}$.

Algorithm 15: Heuristic - Spanning tree interdiction problem

Data: $G = (V, E), w \in \mathbb{N}^{|E|}, c \in \mathbb{R}_+^{|E|}, B \in \mathbb{N}, x^* \in [0; 1]^{|E|}$

1 Let $G^* = (V, E^*)$ such that $E^* = \{e \in E \mid x_e^* > 0\}$;
2 Let $T^w = (V, F^w)$ be the spanning tree minimizing $\sum_{e \in F^w} w_e$ in G^*;
3 $R \leftarrow \emptyset$;
4 $xR \leftarrow 0$;
5 **while** $\sum_{e \in F^w} w_e \leq B$ and $xR < 1$ **do**
6 $\quad e^*, f^* \leftarrow \text{argmax}_{e \in F^w, \delta_e \neq \emptyset} \{(1 - x^*(e))(w_f - w_e) \mid f \in \text{argmin}_{g \in \delta_e} \{w_g\}\}$;
7 $\quad R \leftarrow R \cup \{e^*\}$;
8 $\quad xR \leftarrow xR + x_{e^*}^*$;
9 $\quad F^w \leftarrow F^w \cup \{f^*\} \backslash \{e^*\}$;
10 **end**
11 **if** $\sum_{e \in F^w} w_e > B$ and $xR < 1$ **then**
12 $\quad$ Up-lift the inequality $\sum_{e \in E^* \backslash R} x_e \leq |V| - 2$ using Algorithm 14 and add it to the model;
13 **end**

Algorithm 15 requires $O(|V||E|^2)$ time in the worst case. However, in practice the algorithm is quickly interrupted by the $\sum_{e \in R} x_e^* < 1$ condition of the while loop on line 5.

Example 5

Consider the graph $G = (V, E)$ given by Figure 6.4, with weights on the edges and a budget of 10.

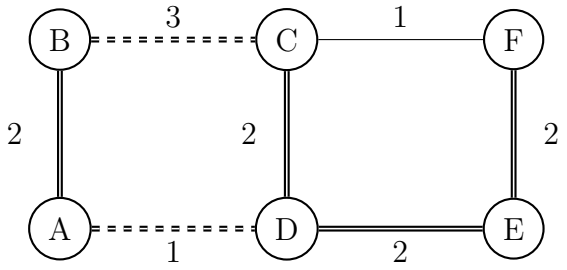


Figure 6.4: An instance of the 1BSTP. The dashed edges correspond to edges with value $x_e^* = \frac{1}{2}$, double edges correspond to edges with value $x_e^* = 1$ and the simple edge has the value $x_e^* = 0$.

Given the fractional solution $x^*_{AB} = x^*_{CD} = x^*_{DE} = x^*_{EF} = 1$, $x^*_{BC} = x^*_{AD} = \frac{1}{2}$ and $x^*_{CF} = 0$, the resulting support graph is given by $G^* = (V, \{AB, BC, CD, AD, DE, EF\})$. Algorithm 15 starts by computing $T^w = (V, \{AB, CD, AD, DE, EF\})$ with a total weight of 9. Then, it performs a swap between AD and BC, increasing the total weight to 11. By deleting $R = \{AD\}$ from E^*, the minimum spanning tree in $G'' = (V, E^* \backslash R)$ violates the budget constraint. Moreover, as $\sum_{e \in R} x_e = \frac{1}{2} < 1$, the valid inequality $x_{AB} + x_{BC} + x_{CD} + x_{DE} + x_{EF} \leq 4$ cuts x^*.

6.3 Valid inequalities obtained via projection

We explore in this section how the projection technique can be exploited to generate valid inequalities at the root of the search tree or during the course of the cutting plane algorithm.

6.3.1 A priori valid inequalities

Due to the arbitrary values of the weights in the budget constraint and the resulting size of our disjunctive formulation, the resolution of (5.17)-(5.22) is not tractable in practice. However, by projecting (5.18)-(5.22) onto the space of the original variables, we manage to generate valid inequalities for the 1BSTP. Let $Proj_x(\mathcal{P}')$ denote the projection of $\mathcal{P}'$ onto the x variables. The following result is a consequence of Proposition 16.

Proposition 25. $Proj_x(\mathcal{P}') = \mathcal{P} = \text{conv}(\mathcal{H})$

The projection cone of $\mathcal{P}'$ is the polyhedral cone $\mathcal{C}$ formed by the vectors $r = (r_{card}, r_E, r_{y_0}, r^{h_1}, r^{h_1}_{y_0}, \ldots, , r^{h_{|Q^*|}}, r^{h_{|Q^*|}}_{y_0})$ satisfying the following system $rD = 0$, where D is the submatrix of the extended formulation associated with variables y^h and y^h_0, $h \in Q^*$, in the extended formulation (5.18)-(5.22):

$$
\begin{pmatrix} r_{card} \\ r_E \\ r_{y_0} \\ r^{h_1} \\ r^{h_1}_{y_0} \\ r^{h_2} \\ r^{h_2}_{y_0} \\ \vdots \\ \vdots \\ r^{h_{|Q^*|}} \\ r^{h_{|Q^*|}}_{y_0} \end{pmatrix}^T
\cdot
\begin{pmatrix}
0_{1,|E|} & \cdots & & \cdots & & \cdots & 0_{1,|E|} & 0 \\
\boldsymbol{-Id_{|E|}} & 0_{|E|,1} & \boldsymbol{-Id_{|E|}} & 0_{|E|,1} & \cdots & \cdots & \boldsymbol{-Id_{|E|}} & 0_{|E|,1} \\
0_{1,|E|} & \mathbf{1} & 0_{1,|E|} & \mathbf{1} & \cdots & \cdots & 0_{1,|E|} & \mathbf{1} \\
\boldsymbol{A} & \boldsymbol{-b^{h_1}} & 0_{l,|E|} & \cdots & & & \cdots & 0_{l,1} \\
0_{1,|E|} & \mathbf{-1} & 0_{1,|E|} & \cdots & & & & 0 \\
\vdots & 0_{l,1} & \boldsymbol{A} & \boldsymbol{-b^{h_2}} & 0_{l,|E|} & \cdots & \cdots & 0_{l,1} \\
 & & 0_{1,|E|} & \mathbf{-1} & 0_{1,|E|} & \cdots & & 0 \\
 & & \vdots & 0_{l,1} & \ddots & \ddots & & \vdots \\
 & & & \vdots & & \ddots & \ddots & 0_{1,|E|} \\
 & & & & & & \boldsymbol{A} & \boldsymbol{-b^{h_{|Q^*|}}} \\
0_{1,|E|} & 0 & 0_{1,|E|} & 0 & \cdots & \cdots & 0_{1,|E|} & \mathbf{-1}
\end{pmatrix}
= 0^{1\times|Q^*|(|E|+1)},
$$

where $r_{card}, r_{y_0} \in \mathbb{R}$, $r_E \in \mathbb{R}^{|E|}$, $r^h \in \mathbb{R}^l_+$ and $r^h_{y_0} \in \mathbb{R}_+$ for $h \in Q^*$.

By construction, the components r_E and r_{y_0} are associated with the block containing identity submatrices for each polyhedron $P_h,\ h \in Q^*$ in the disjunction. Components r^h and $r^h_{y_0}$, $h \in Q^*$, are associated with a unique block formed by the submatrix A, the vector $-b^h$ and a coefficient -1. Due to the matrix structure, we obtain the following relations between the components:

Proposition 26. *Any ray r of the polyhedral cone $\mathcal{C}$ satisfies:*

$$r_E = r^h A, \qquad \forall h \in Q^*, \tag{6.2}$$
$$r_{y_0} = r^h b^h + r^h_{y_0}, \qquad \forall h \in Q^*, \tag{6.3}$$
$$r_{y_0} \geq r^h b^h, \qquad \forall h \in Q^*. \tag{6.4}$$

Proof. Equations (6.2) and (6.3) stem from the systems $-r_E + r^h A = 0_{1,|E|}$ and $r_{y_0} - r^h b^h - r^h_{y_0} = 0$ for all $h \in Q^*$. Inequalities (6.4) follow from (6.3) and the non negativity of components $r^h_{y_0}$. □

By exploiting the structure of matrix D, we deduce the following results:

Proposition 27. *The matrix D of the projection cone has a full column rank.*

Proof. In each polyhedron $P_h,\ h \in Q^*$, of the disjunction, the non-negativity of variables $y^h_e,\ e \in E$ and y^h_0 is imposed. Therefore the lines of the matrix cone D associated to each polyhedron P_h of the disjunction have the following form:

$$\begin{pmatrix} 0 & \cdots & 0 & -1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & 0 & -1 & \cdots & 0 & 0 & \vdots & & \vdots \\ \vdots & & \vdots & \vdots & 0 & \ddots & 0 & \vdots & \vdots & & \vdots \\ \vdots & & \vdots & 0 & 0 & \cdots & -1 & 0 & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \end{pmatrix}$$

y^h y_0^h

This shows that no column of matrix D is in the space of all other columns. □

Note that the projection cone $\mathcal{C}$ is not pointed since it contains a line with direction vector $(1, 0, \ldots, 0)$.

The particular structure of the projection cone influences the form of the resulting valid inequalities obtained by projection. Using the Balas's projection theorem [8], we obtain the following description of $Proj_x(\mathcal{P}')$ that exploits the rays of $\mathcal{C}$:

$$Proj_x(\mathcal{P}') = \{x \in \mathbb{R}^{|E|} \mid r \begin{pmatrix} 1_{1,|E|} \\ Id_{|E|} \\ O_{(l+1)|Q^*|+1,|E|} \end{pmatrix} x \leq r \begin{pmatrix} |V|-1 \\ 0_{|E|} \\ 1 \\ O_{(l+1)|Q^*|} \end{pmatrix}, r \in \mathcal{C}\}. \tag{6.5}$$

where

$$\begin{pmatrix} 1_{1,|E|} \\ Id_{|E|} \\ O_{(l+1)|Q^*|+1,|E|} \end{pmatrix}$$

is the submatrix related to the design variables in the extended formulation (5.18)-(5.22), and

$$\begin{pmatrix} |V|-1 \\ 0_{|E|} \\ 1 \\ O_{(l+1)|Q^*|} \end{pmatrix}$$

is the right-hand side of the extended formulation (5.18)-(5.22).

Proposition 28. *Any ray r of the polyhedral cone $\mathcal{C}$ induces a valid inequality*

$$r_E x \leq r_{y_0} \tag{6.6}$$

and is valid for the 1BSTP.

Proof. Exploiting 6.5 with a ray r of the projection cone $\mathcal{C}$, we obtain the inequality $(r_{card} + r_E)x \leq r_{card}(|V|-1) + r_{y_0}$. As $\sum_{e \in E} x_e = |V| - 1$, we can simplify the inequality to the desired form. □

Due to its exponential size, building a ray that is a generator of $\mathcal{C}$ is a problem that does not belong to NP. Moreover, not all the generators of the projection cone induce facets of $Proj_x(\mathcal{P}')$ [8]. Since computing generators is hard, we build particular rays that combine both the spanning tree and the knapsack structures. Although one can fix any vector $r_E \in \mathbb{R}^{|E|}$, the results of Proposition 29 reduce the good candidates to non-negative vectors r_E.

Given a ray r of $\mathcal{C}$, we denote by $S_r^+ = \{e \in E \mid r_{E,e} > 0\}$ and $S_r^- = \{e \in E \mid r_{E,e} < 0\}$ the set of edges associated with a non negative or a non positive coefficient in r_E, where $r_{E,e}$ is the component of r_E associated with edge e. A particular case of vector r_E occurs when $|S_r^+ \cup S_r^-| = 1$. In the latter case, if $|S_r^+| = 1$ (respectively $|S_r^-| = 1$) and if the related edge e is neither a forbidden or a necessary edge, the resulting inequality $r_E x \leq r_{y_0}$ is redundant (or corresponds) with the trivial inequalities $x_e \leq 1$ (respectively $x_e \geq 0$). The following proposition considers more complex rays.

Proposition 29. *A necessary condition for a ray r such that $|S_r^+ \cup S_r^-| > 1$ of $\mathcal{C}$ to be a generator is that $r_E \in \mathbb{R}_+^{|E|}$.*

Proof. Consider a ray $r = (r_{card}, r_E, r_{y_0}, r^{h_1}, r_{y_0}^{h_1}, \ldots, r^{h_{|Q^*|}}, r_{y_0}^{h_{|Q^*|}})$ such that $|S_r^+ \cup S_r^-| > 1$ and $|S_r^-| > 0$ and suppose that it is a generator of $\mathcal{C}$. By (6.2), $r_E = r^h A,\ \forall h \in Q^*$. In particular, any component $r_{E,e}$ satisfies $r_{E,e} = r^h A^e$ where A^e is the column of matrix A related to edge e. Recall that matrix A is associated with constraints (5.4)-(5.6). Therefore, each component of A^e is either 0 or 1 except the one associated to the non negativity of variable y_e^h, i.e. $-y_e^h \leq 0$, which is equal to -1. Consequently, $r_{E,e} = \sum_{i \neq e} r_i^h A_i^e - r_e^h$, where the summation is non negative since all the components r_i^h and $A_i^e \geq 0,\ i \neq e$. This implies

$$r_{E,e} + r_e^h \geq 0. \tag{6.7}$$

Consider a vector $r' = (r'_{card}, r'_E, r'_{y_0}, r'^{\ h_1}, r'^{\ h_1}_{y_0}, \ldots, , r'^{\ h_{|Q^*|}}, r'^{\ h_{|Q^*|}}_{y_0})$ corresponding to a copy of r, except that $r'_{E,e} = 0$ for every $e \in S_r^-$ and $r'^{\ h}_e = r_e^h + r_{E,e}$ for every edge $e \in S_r^-$ and $h \in Q^*$, where the components r_e^h and $r'^{\ h}_e$ of vectors r^h and $r'^{\ h}$ are the components of vectors r^h and $r'^{\ h}$ associated with constraints $-y_e^h \leq 0$ of the system $Ax \leq b^h,\ h \in Q^*$. Vector r' satisfies conditions (6.2), (6.3) and (6.4). Hence, r' is a ray of $\mathcal{C}$. Moreover, by 6.7 we have $r'_E \geq 0$. Now, consider vectors $r^e = (r^e_{card}, r^e_E, r^e_{y_0}, r^{e\ h_1}, r^{e\ h_1}_{y_0}, \ldots, , r^{e\ h_{|Q^*|}}, r^{e\ h_{|Q^*|}}_{y_0})$, $\forall e \in E$, such that r^e is a vector with zero components except for components $r^e_{E,e} = -1$ and $r^{e\ h}_e = 1,\ h \in Q^*$. Vectors $r^e,\ e \in E$ satisfy conditions (6.2), (6.3) and (6.4), hence they correspond to rays of $\mathcal{C}$. By construction, we have $r = r' + \sum_{e \in S_r^-} |r_{E,e}| \cdot r^e$, contradicting the extremality of r. □

Note that Proposition 29 implies that every valid inequality $r_E x \leq r_{y_0}$ such that $|S_r^+ \cup S_r^-| > 1$ and $r_E \notin \mathbb{R}_+^{|E|}$ is not a facet.

According to Propositions 28 and 29, it suffices to select a good candidate $r_E \in \mathbb{R}_+^{|E|}$ component and compute the tightest right-hand side r_{y_0}. By Proposition 25, only the choices of r_E and r_{y_0} such that there exists a ray $r = (r_{card}, r_E, r_{y_0}, r^{h_1}, r_{y_0}^{h_1}, \ldots, , r^{h_{|Q^*|}}, r_{y_0}^{h_{|Q^*|}})$ of $\mathcal{C}$ are interesting. Starting from an arbitrary component $r_E \in \mathbb{R}_+^{|E|}$, we give an associated ray $r = (r_{card}, r_E, r_{y_0}, r^{h_1}, r_{y_0}^{h_1}, \ldots, , r^{h_{|Q^*|}}, r_{y_0}^{h_{|Q^*|}})$ that belongs to $\mathcal{C}$ and such that the valid inequality $r_E x \leq r_{y_0}$ defines a proper face of $\mathcal{P}$. We assume that $||r_E||_{+\infty} < +\infty$.

First, we compute the best coefficients $r^h,\ h \in Q^*$. For every polyhedron $P_h = \{x \in \mathbb{R}^{|E|} \mid Ax \leq b^h\},\ h \in Q^*$ in the disjunction, consider the following linear program:

$$\max\ \{r_E x \mid Ax \leq b^h\}. \tag{6.8}$$

Since $h \in Q^*$, the feasible domain is non empty and corresponds to the intersection of the graphic and the partition matroids. Therefore, the above linear program has an integer optimal solution. By $||r_E||_{+\infty} < +\infty$, problem (6.8) has a finite optimum. Therefore, the dual problem is well defined and the component r^h is an optimal solution.

$$\min\ \{r^h b^h \mid r^h A = r^E,\ r^h \geq 0\}. \tag{6.9}$$

We now focus on the component r_{y0} which will define the right-hand side of the valid inequality associated with our constructed ray r. By (6.4), $r_{y0} \geq \max_{h\in Q^*}\{r^h b^h \mid r^h$ optimal solution of (6.9)$\}$. The following proposition shows that the strongest inequality is obtained by setting $r_{y0} = \max_{h\in Q^*}\{\ r^h b^h \mid r^h$ optimal solution of (6.9)$\}$.

Proposition 30. *The hyperplane $\{x \in \mathbb{R}^{|E|} \mid r_E x = r_{y_0}\}$ is a proper face of $\mathcal{P}$, where $r_{y0} = \max_{h\in Q^*}\{r^h b^h \mid r^h$ optimal solution of (6.9)$\}$.*

Proof. Consider $h^* \in \arg\max_{h\in Q^*}\{r^h b^h \mid r^h$ optimal solution of (6.9)$\}$ and the linear program associated with the polyhedron P_{h^*} in the disjunction. By the strong duality theorem, $r^{h^*} b^{h^*} = r_E x(F^*)$ where $T^* = (V, F^*)$ is an optimal spanning tree of the optimization problem (6.8) related to P_{h^*}. By definition, we get $r_E x(F^*) = r_{y_0}$. □

Finally, by setting $r^h_{y_0} = r_{y_0} - r^h b^h,\ h \in Q^*$, we complete the construction of our ray.

Computing the lowest right-hand side r_{y_0} requires handling explicitly all polyhedra in Q^*. Instead, one could solve the following 1BSTP and compute the tightest right-hand side of the valid inequality. For this purpose, consider the program:

$$Z = \max\ r_E x \tag{6.10}$$

$$\sum_{e\in E} x_e = |V| - 1, \tag{6.11}$$

$$\sum_{e\in E[S]} x_e \leq |S| - 1, \qquad \forall S \subset V, 2 < |S| < |V| - 1, \tag{6.12}$$

$$\sum_{e\in E} w_e x_e \leq B, \tag{6.13}$$

$$x_e \in \{0, 1\}, \qquad \forall e \in E. \tag{6.14}$$

We have the following result:

Proposition 31. $Z = r_{y_0} = \max_{h\in Q^*}\{r^h b^h \mid r^h$ optimal solution of (6.9)$\}$.

Proof. Consider $h^* \in \arg\max_{h\in Q^*}\{r^h b^h \mid r^h$ optimal solution of (6.9)$\}$, let T^{h^*} denotes an optimal solution of the linear program (6.8) associated with polyhedron P_{h^*} in the disjunction. Let T^* denote an optimal solution of (6.10)-(6.14). Since T^{h^*} is a feasible solution of (6.10) – (6.14), we have $Z \geq r^{h^*} b^{h^*}$. Define b^* the right-hand side vector of the

matroid partitions in the polytope P_{b^*} such that $x(T^*) \in P_{b^*}$. By definition, $x(T^*)$ is an optimal solution of P_{b^*}. This implies that $Z = r_E x(T^*) = r^* b^{b^*}$, where the last equation follows from the strong duality theorem applied to problems (6.8) and (6.9) for $b^h = b^*$. The result follows since $r^{b^*} b^{b^*} \leq r^{h^*} b^{h^*} = r_{y0}$, where r^{b^*} is an optimal solution of the dual problem (6.9) associated with P_{b^*}. This shows the claimed result. □

In the particular case where a ray of $\mathcal{C}$ has only binary coefficients, problem (6.10)-(6.14) can be solved in polynomial time using Lagrangian relaxation technique as presented in Section 4.2.2.

By combining Propositions 28 and 31, solving (6.10)-(6.14) for a given vector r_E is sufficient to generate the valid inequality $r_E x \leq r_{y_0}$ associated with the exponential size ray $r = (r_{card}, r_E, r_{y_0}, r^{h_1}, r^{h_1}_{y_0}, \ldots, , r^{h_{|Q^*|}}, r^{h_{|Q^*|}}_{y_0})$ without knowing all of its coefficients. In the following, we explore different choices of vectors $r_E \geq 0$ yielding effective classes of valid inequalities.

A judicious choice of vector $r_E \geq 0$ can be generated by exploiting the structure of the polyhedra $P_h,\ h \in Q^*$ in the disjunction of the exact extended formulation (5.18)-(5.22). By construction, every ray r satisfies $r_E = r^h A,\ \forall h \in Q^*$, hence one could choose a vector r_E that is a positive combination of the left-hand side of partition, subtour or trivial inequalities in $P_h,\ h \in Q^*$. Note that if the latter combination is obtained by only subtour of trivial inequalities, the resulting valid inequality may be redundant with the subtour formulation. More interesting valid inequalities are obtained by considering at least one partition inequality in the combination.

Example 6

Consider the graph given by Figure 6.5 with a budget 5.

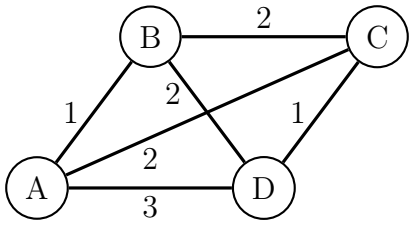


Figure 6.5: An instance of the 1BSTP.

The related disjunctive formulation has been given in Example 3. The associated projection cone $\mathcal{C}$ has 1864 generators (instead of 34167828 without removing the redundant subtour and trivial inequalities). 950 generators lead to non-redundant inequalities with the trivial inequalities and only 13 are associated with facet-defining inequalities.

An extreme point of the 1-budgeted subtour formulation is given by $x_{AB} = 0.5,\ x_{AC} = 0,\ x_{AD} = 0.5,\ x_{BC} = 0,\ x_{BD} = 1$ and $x_{CD} = 1$. This solution is cut by the valid inequality $x_{AD} + x_{BD} + x_{BC} + x_{CD} \leq 2$ obtained by projecting the ray r, where $r_E = (0,0,1,1,1,1), r_{y0} = 2,\ r_{card} = 0, r^{(2,1,0)} = (0,1,1,0,0,1,0,0,0,0,0,1), r^{(2,1,0)}_{y_0} = 0, r^{(2,0,1)} = (0,1,1,0,0,1,0,0,0,0,0,1), r^{(2,0,1)}_{y_0} = 0, r^{(1,2,0)} = (0,0,1,1,0,0,0,0,0,0,0,0,0,0,0)$ and $r^{(1,2,0)}_{y_0} = 0$. The combinations $r_E = r^{(2,1,0)}A$ and $r_E = r^{(2,0,1)}A$ are obtained with the partition inequalities related to $E_2 = \{AC, BC, BD\},\ E_3 = \{AD\}$ and the trivial inequalities

$-x_{AC} \leq 0$ and $x_{CD} \leq 1$ for $P_{(2,1,0)}$ and $P_{(2,0,1)}$. The combination $r_E = r^{(1,2,0)}A$ is obtained with the subtour constraint $x_{BC} + x_{BD} + x_{CD} \leq 2$ and the partition inequality $x_{AD} \leq 0$. The resulting inequality considers both the knapsack and the spanning tree problems.

In chapter 7 we give an algorithm that generates efficient vectors r_E exploiting the geometry of the Lagrangian relaxation and partition inequalities. In the following, we investigate classes of vectors r_E that generate valid inequalities that dominate the cover inequalities.

Let $T = (V, F)$ be an infeasible spanning tree. Algorithm 16 builds a vector r_E leading to an inequality $r_E x \leq r_{y_0}$ that dominates the cover inequality $\sum_{e \in F} x_e \leq |V| - 2$. The algorithm considers a subgraph $G' = (V, S)$ of G such that S contains all edges involved in partitions $E_1, \ldots, E_k$ with $E_i \cap F \neq \emptyset,\ i = 1, \ldots, k$. The algorithm removes edges in $S \backslash F$ such that at each iteration the increase in total weight of a MST in G' is maximum until there is no feasible solution in G'. The incidence vector of the last edge set of G' defines the vector r_E. By computing the tightest value of r_{y_0}, the resulting inequality may have a better right-hand side than the cover inequality.

Algorithm 16: Generation of an inequality $r_E x \leq r_{y_0}$ that dominates a cover inequality

Data: $G = (V, E), w \in \mathbb{N}^{|E|}, c \in \mathbb{R}_+^{|E|}, B \in \mathbb{N}, T = (V, F)$ an infeasible spanning tree

Let $G' = (V, S)$ such that $S = \{e \in E \mid w_e = w_f, f \in F\}$;
Let $T^w = (V, F^w)$ be the spanning tree minimizing $\sum_{e \in F^w} w_e$ in G';
$R \leftarrow \emptyset$;
while $\sum_{e \in F^w} w_e \leq B$ **do**
  $e^*, f^* \in \operatorname{argmax}_{e \in F^w \backslash F}\{(w_f - w_e \mid f \in \operatorname{argmin}_{g \in \delta_e^{S \backslash R}}\{w_g\}\}$;
  $R \leftarrow R \cup \{e^*\}$;
  $F^w \leftarrow F^w \cup \{f^*\} \backslash \{e^*\}$;
end
Let r_E be the incidence vector of $S \backslash R$;
Compute r_{y_0} by solving (6.10) – (6.14);

We denote $\delta_e^{S \backslash R} = \{uv \in S \backslash R \mid u \in W, v \in V \backslash W\}$ where W is the node set of one of the two components created by removing e from T^w. Note that as $F \subset S \backslash R$, Algorithm 16 stops and $r_{y_0} \leq |V| - 2$ as there is no feasible solution in G'.

Proposition 32. *The inequality $r_E x \leq r_{y_0}$, where r_E is obtained by Algorithm 16, dominates the cover inequality $\sum_{e \in F} x_e \leq |V| - 2$.*

Proof. By construction, we have $F \subset S \backslash R$ and by Proposition 24, $r_{y_0} \leq |V| - 2$. □

6.3.2 Cutting plane approach

Our integer extended formulation could also be used in order to derive valid inequalities cutting off fractional solutions generated in the Branch-and-Cut algorithm. Consider a

fractional solution $x^* \notin \mathcal{P}$. Since $x^* \notin Proj_x(\mathcal{P}')$, then the following linear program is infeasible:

$$\max\ 0y^h \tag{6.15}$$
$$\sum_{h \in Q^*} y^h = x^*, \tag{6.16}$$
$$Ay^h - b^h y_0^h \leq 0, \qquad \forall h \in Q^*, \tag{6.17}$$
$$\sum_{h \in Q^*} y_0^h = 1, \tag{6.18}$$
$$y^h, y_0^h \geq 0, \qquad \forall h \in Q^*. \tag{6.19}$$

The dual problem:

$$\min\ x^*\alpha + \gamma \tag{6.20}$$
$$\alpha + \beta^h A \geq 0, \qquad \forall h \in Q^*, \tag{6.21}$$
$$-\beta^h b^h + \gamma \geq 0, \qquad \forall h \in Q^*, \tag{6.22}$$
$$\beta^h \geq 0, \qquad \forall h \in Q^*. \tag{6.23}$$

where

- $\alpha \in \mathbb{R}^{|E|}$ is the vector of dual variables associated with constraints (6.16),
- $\beta^h \in \mathbb{R}^m_+$ is the vector of dual variables associated with constraints (6.17) for a given $h \in Q^*$, and
- $\gamma \in \mathbb{R}$ is the dual variable associated with constraint (6.18).

The dual problem has a feasible solution given by the zero vector, and thus it is unbounded. Let $\mathcal{D}'$ denote the set of vectors (α, β, γ) satisfying constraints (6.21)-(6.23). By using the same argument as in Proposition 27, one can show that the matrix of the system (6.16)-(6.19) has a full rank. Therefore, the polyhedron defined by (6.21)-(6.23) has an extreme ray $(\alpha^*, \beta^{*1}, \ldots, \beta^{*|Q^*|}, \gamma^*)$ such that $\alpha^* x + \gamma^* < 0$. Observe that any solution (α, β, γ) defines a valid inequality $\alpha x \geq -\gamma$ for the convex hull of the disjoint $\bigcup_{h \in Q^*} P_h$, [8]. Consider now the following bounded linear program:

$$\min\ x^*\alpha + \gamma \tag{6.24}$$
$$\alpha + \beta^h A \geq 0, \qquad \forall h \in Q^*, \tag{6.25}$$
$$-\beta^h b^h + \gamma \geq 0, \qquad \forall h \in Q^*, \tag{6.26}$$
$$x^*\alpha + \gamma \geq -\epsilon, \tag{6.27}$$
$$\beta^h \geq 0, \qquad \forall h \in Q^*. \tag{6.28}$$

where $\epsilon > 0$ is a given scalar.

This linear program has an optimal extreme point solution. This optimal extreme point is a point $(\alpha^*, \beta^*, \gamma^*) \in \mathcal{D}'$ such that the set of tight constraints except (6.27) is of $|Q^*|(|E|+1)-1$. Therefore, $(\alpha^*, \beta^*, \gamma^*)$ is an extreme ray of $\mathcal{D}'$. The following proposition shows that it yields a valid inequality cutting off x^*.

Proposition 33. *The inequality $\alpha x+\gamma \geq 0$ is valid for $Proj_x(\mathcal{P})$ and cuts off the fractional solution x^*.*

Proof. The primal and dual problems (6.15)-(6.19) and (6.20)-(6.23) associated with any feasible solution $\overline{x} \in Proj_x(\mathcal{P})$ are feasible and have a zero optimal value. Observe that $(\alpha^*, \beta^{*1}, \ldots, \beta^{*|Q^*|}, \gamma^*)$ satisfies constraints (6.21)-(6.22) of the dual problem associated with $\overline{x}$. Therefore, $\alpha^*\overline{x} + \gamma^* \geq \min_{(\alpha,\beta,\gamma)\ satisfies\ (6.21)-(6.23)} \alpha\overline{x} + \gamma = 0$. □

The following construction shows that the vector $(\alpha^*, \beta^{*1}, \ldots, \beta^{*|Q^*|}, \gamma^*)$ corresponds to a generator of the projection cone $\mathcal{C}$. Let $r \in \mathbb{R}^{|E|+2+|Q^*|(l+1)}$ such that $r_{card} = 0$, $r_E = -\alpha^*$, $r_{y_0} = \gamma^*, r^h = \beta^{*h}$ and $r^h_{y_0} = \gamma^* - \beta^{*h}b^h$ for $h \in Q^*$, we have $\alpha^* = \beta^{*h}A$,$\forall h \in Q^*$ by optimality of $(\alpha^*, \beta^{*1}, \ldots, \beta^{*|Q^*|}, \gamma^*)$ and $\gamma^* = \beta^{*h}b^h + \gamma^* - \beta^{*h}b^h,\ \forall h \in Q^*$. Therefore, r is a ray of the projection cone $\mathcal{C}$. Those inequalities contrast with the valid inequalities discussed at the end of Section 6.3.1 that do not necessarily correspond to generators of $\mathcal{C}$. Recall that any facet of the convex hull of $\mathcal{H}$ is a generator of the projection cone $\mathcal{C}$ but the converse is not true.

Due to the size of the extended formulation, it is not possible to exploit it within a Branch-and-Cut algorithm. Based on the structure of a fractional solution x^*, we propose a new extended formulation based on the disjunction of a tractable number of polyhedra $P'_1, \ldots, P'_l$. The price to pay is that the latter is not exact. The goal is to efficiently cut off x^* via projection. We propose next a general procedure to build the designed disjunction. Given a fractional solution $x \notin \mathcal{P}$, the polyhedra $P'_1, \cdots, P'_l$ must satisfy the following conditions:

- $\mathcal{P} \subseteq conv(\bigcup_{i=1}^{l} P'_i)$,
- $x^* \notin conv(\bigcup_{i=1}^{l} P'_i)$.

The second condition can be easily fulfilled by considering any valid inequality, for instance a subtour or a cover inequality, violated by x^* in all the polyhedra in the disjunction.

For instance, one could consider the following disjunction based on two polyhedra:

$$P'_1 = \begin{cases} x(S) \leq \lfloor x^*(S) \rfloor \\ C^1 x \leq d_1 \\ x \geq 0 \end{cases} \qquad P'_2 = \begin{cases} x(E\backslash S) \leq \lfloor x^*(E\backslash S) \rfloor \\ C^2 x \leq d_2 \\ x \geq 0 \end{cases}$$

where $S = \bigcup_{i=j}^{k} E_i$ a union of sets of the partition associated with the highest weights $w_j, \ldots, w_k$ and where $C^i x \leq d_i, i = 1, 2$ are 1BSTP's inequalities. The disjunction is illustrated in Figure 6.6.

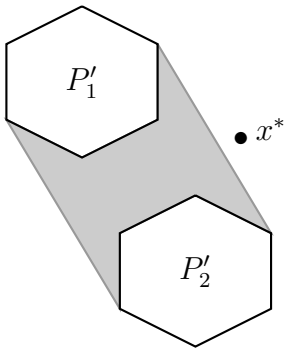


Figure 6.6: Disjunction of two polyhedra and a fractional solution x^*. Every solution belonging either in one of the two polytopes or in the gray area is a feasible solution of the disjunctive problem.

The resulting valid inequalities obtained via projection may be strong. However, experimental results show that implementing this idea is time consuming. An interesting line of research is to use the projection technique to cut off fractional solutions.

Chapter 7

Branch-and-Cut for the minimum 1-budgeted spanning tree problem and computational experiments

In this chapter, we present the essential components of a Branch-and-Cut algorithm and analyse its performance. We begin with preprocessing and probing techniques. Different basic techniques are discussed in the literature to improve the representation of an integer program and fix binary variables [69]. We present new and simple algorithms that exploit the combinatorial structure of the STP. Next, we summarize all cutting planes in the algorithm, including at the root node. Most of the separation algorithms and heuristics are described in detail in Chapter 6. However, implementing all the ideas discussed in Chapter 6 is time consuming. We leave the practical investigations of the remaining valid inequalities obtained by projection as a line of future research. There may be different choices for executing each component of the algorithm and testing all of them would be prohibitive. Instead, we describe for each of these components the best choice and present computational experiments. Our goal is to demonstrate that our algorithm outperforms generic Branch-and-Cut and existing algorithms. All the computational experiments were performed on a Dell desktop Intel Core^{TM} i7, with 1.90GHz, 4 physical cores, 8 logical processors and using up to 8 threads. All instances were solved using Gurobi solver. A sufficient time limit of 3600 seconds was imposed on the resolution of each instance. The MIP Gap precision has been set to 10^{-7}. The source code was written in Julia 1.3.1, using JuMP, Graphs and Gurobi as main libraries.

7.1 Problem classes

The difficulty of an instance of the 1BSTP depends on the topology of the graph (the spanning tree part) and the structure of the costs and weights (the knapsack part). We investigate 3 classes of graphs with increasing density:

- *Grid*: A rectangular $p \times q$ grid graph, where $p = \left\lceil \frac{|V|}{\sqrt{|V|}} \right\rceil$ and $q = \lfloor \sqrt{|V|} \rfloor$.
- *Geom*: Random geometric instances based on the construction given by Johnson et al. [42] for the prize collecting steiner tree problem. In these instances, $|V|$ vertices

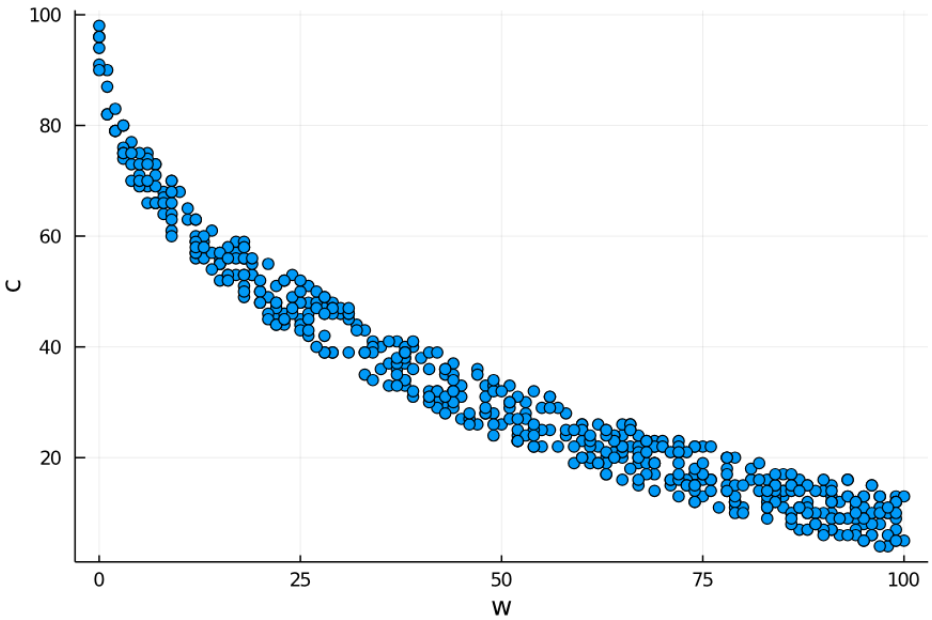
are randomly distributed in a unit square. An edge is added for every pair of nodes u and v if the Euclidian distance between them is less than $\frac{1.8}{\sqrt{|V|}}$. If necessary, we add edges with minimum distances to obtain a connected graph.

- *Rand*: Random dense instances where an edge is added with a uniform probability of 0.7 between any pair of nodes.

For all the 3 classes of graphs, we consider 2 classes of cost and weight values:

- R: For each edge e, both the weight w_e and the cost c_e values are independently and uniformly distributed over [0, 100].
- I: For each edge e, the weight w_e is uniformly distributed over $[0; 100]$ and we set the cost value to $c_e = 90 + a - \lfloor 0.5 \times \sqrt{40000 - (w_e - 200)^2} \rfloor$, where a is a uniform random value in $[0, 10]$.

In instances R, the weights w_e and the costs c_e are uncorrelated. However, in real world situations, the criteria corresponding to the cost and the weight functions are, in general, conflicting. Instances I can be expected to be the most difficult. Moreover, the latter are based on Pisinger's circle instances [61]. The following picture illustrates the contrast between the costs and the weights of the I instances.



For all these instances, we set the budget $B = 30(|V| - 1)$.

7.2 Benchmark algorithms

In the experiment part, we will compare the resolution time of Algorithm 28 with two efficient algorithms. The first is a generic Branch-and-Cut Algorithm 17 based on the subtour formulation. Starting with an empty set of subtour constraints, the algorithm

generates them dynamically using the separation algorithm [59]. The latter requires $|V|-2$ flows computation in order to find a most violated inequality. In order to improve the running time, the separation Algorithm 18 adds to the model all the violated inequality encountered during the course of the flows. Computational experiments show that a good upper bound on the total number on added subtour inequalities is around $40|V|$. The whole algorithm already outperforms most of the existing ones, see Appendices 2, 3 and 4.

Algorithm 17: Branch-and-Cut based on subtour formulation

Data: $G = (V, E), w \in \mathbb{N}^{|E|}, c \in \mathbb{R}_+^{|E|}, B \in \mathbb{N}$

1 Build the Integer Linear Problem:
$\min\{\sum_{e\in E} c_e x_e \mid \sum_{e\in E} x_e = |V| - 1, \sum_{e\in E} w_e x_e \leq B, x \in \mathbb{B}^{|E|}\}$;

2 Solve the Branch-and-Cut algorithm with separation algorithm 18 ;

Algorithm 18: Separation of subtour inequalities

Data: $G = (V, E)$, $x^* \in [0:1]^{|E|}, K \in \mathbb{N}$

1 Consider the support graph $G^* = (V, E^*)$ where $E^* = \{e \in E \mid x^*(e) > 0\}$;
2 **if** x^* is integer **then**
3 **if** there are $p > 1$ connected components in G^* **then**
4 **for** every connected component $G^*[V_i]$, $i = 1, \ldots, p$ **do**
5 **if** component $G^*[V_i]$ contains more than $|V_i| - 1$ edges **then**
6 Add the subtour constraint $\sum_{e\in E[V_i]} x_e \leq |V_i| - 1$;
7 **end**
8 **end**
9 **end**
10 **else**
11 **if** the total number of added inequalities is lower than K **then**
12 **if** there are $p > 1$ connected components in G^* **then**
13 **for** every connected component $G^*[V_i]$, $i = 1, \ldots, p$ **do**
14 **if** $\sum_{e\in E[V_i]} x_e > |V_i| - 1$ **then**
15 Add the subtour constraint $\sum_{e\in E[V_i]} x_e \leq |V_i| - 1$;
16 **end**
17 **end**
18 **else**
19 Run Algorithm 3(G^*);
20 Run an adaptation of Padberg and Wolsey's separation algorithm [59] in which for every maximum flow computed, the obtained subtour inequality is added to the model if it is violated;
21 **end**
22 **end**
23 **end**

Next, we consider a Branch-and-Bound algorithm based on the compact MTZ formulation (3.14)-(3.20). The latter is enhanced with the following family of valid inequalities, which prohibits to use the same edge in the two directions:

$$x_{ij} + x_{ji} \leq 1, \qquad \{i,j\} \in E. \tag{7.1}$$

This formulation appears to be the most efficient among the compact ones, see Appendix 5.

In all the experiments with the benchmark algorithms, we solved the problems with the default solver's parameter setting. In order to find effectively an optimal solution, the solver implements sophisticated preprocessing techniques and adds cuts. In contrast, all these options are disabled in the experimentation of Algorithm 28.

7.3 Preprocessing and probing procedures

A key component of the Branch-and-Cut algorithm is tightening the formulation so as to make the difference in the objective function values between the linear program and the integer program as small as possible. There are various ways to achieve this goal. Probing techniques tentatively set a binary variable x_e to either 0 or 1 and investigate the consequences in terms of the feasibility of the problem. Constraint generation aims to add valid inequalities either corresponding to inequalities of the STP or the KP or valid inequalities obtained from the extended formulation of the 1BSTP. All these routines are implemented at the root node. In addition, we provide an initial feasible solution (MIP Start) to the solver in order to reduce the overall solve time.

7.3.1 Local search method

We discussed in Chapter 4 an efficient algorithm based on the Lagrangian relaxation to yield a 2-approximated solution. Although the computed solution is an optimal solution of the Lagrangian relaxation (4.8)-(4.11), it may be improved into a solution with a smaller cost. Local search methods are heuristics exploring the neighborhood of a solution in order to find better ones [52]. Considering the 1BSTP, Algorithm 19 performs swaps such that at each iteration the increase in total weight is minimum. At the end of the algorithm, we supply the solution to the solver as a MIP Start to potentially eliminate portions of the search space and result in a smaller Branch-and-Cut tree.

Algorithm 19: Local search method

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, $T = (V, F)$ a feasible spanning tree

do
  $e^*, f^* \leftarrow \min\{w_e - w_f \mid e \in E \backslash F, f \in F, c_e < c_f, \sum_{h \in F \cup \{e\} \backslash \{f\}} w_h \leq B, f \text{ in the cycle created by adding } e \text{ to } T\}$;
  if $e^* \neq \emptyset$ and $f^* \neq \emptyset$ **then**
    $F \leftarrow F \cup \{e\} \backslash \{f\}$;
  end
while $e^* \neq \emptyset$ and $f^* \neq \emptyset$;
Returns $T = (V, F)$;

Tables 7.1 and 7.2 show the quality of the approximate solutions obtained by combining Algorithm 4 and Algorithm 19. We compare the combination of our algorithms with the 2-approximation algorithm of Yamada et al. [82] and with Algorithm 5. The latter is the first phase of both other algorithms, we consider the feasible solution obtained at the upper bound with $\epsilon = 10^{-3}$. All these algorithms start with the initial search interval $[0; 1]$. Gaps are computed by the formula $\frac{\sum_{e \in E} c_e x'_e}{\sum_{e \in E} c_e x^*_e} - 1$ where x' is the incidence vector of the approximate solution and x^* is an optimal solution.

Instance	$\|V\|$	Gap (%) Alg. 4+19	Time (s) Alg. 4+19	Gap (%) Alg. [82]	Time (s) Alg. [82]	Gap (%) Alg. 5
I_{Grid}	1000	0.0200	0.074	**0.0155**	0.053	0.0421
I_{Grid}	1000	**0.0378**	0.223	0.0600	0.019	0.1423
I_{Grid}	1000	**0.0223**	0.084	0.0357	0.019	0.0713
I_{Grid}	1000	**0.0067**	0.035	0.0736	0.019	0.1895
I_{Grid}	1000	**0.0112**	0.034	0.8296	0.013	1.9229
I_{Grid}	1500	**0.0236**	0.311	0.0634	0.038	0.1460
I_{Grid}	1500	**0.0044**	0.143	0.0340	0.035	0.0577
I_{Grid}	1500	**0.0148**	0.320	0.0251	0.033	0.0606
I_{Grid}	1500	**0.0250**	0.243	0.0367	0.033	0.0602
I_{Grid}	1500	0.0074	0.159	**0.0059**	0.037	0.0222
I_{Geom}	700	0.0345	0.198	**0.0034**	0.037	0.0896
I_{Geom}	700	**0.0069**	0.097	**0.0069**	0.034	0.0138
I_{Geom}	700	**0.0035**	0.133	0.0484	0.033	0.1037
I_{Geom}	700	0.0414	0.222	**0.0380**	0.034	0.0829
I_{Geom}	700	**0.0242**	0.181	0.0346	0.030	0.0795
I_{Geom}	1300	**0.0149**	0.409	**0.0149**	0.157	0.0204
I_{Geom}	1300	**0.0111**	0.363	0.0334	0.097	0.0761
I_{Geom}	1300	**0.0093**	0.298	0.0670	0.093	0.1209
I_{Geom}	1300	**0.0205**	0.571	0.0560	0.099	0.1362
I_{Geom}	1300	0.0280	0.636	**0.0262**	0.124	0.0598
I_{Rand}	300	**0 (opt)**	1.513	0.0448	0.423	0.3136
I_{Rand}	300	**0 (opt)**	1.178	0.0090	0.464	0.0897
I_{Rand}	300	**0.0090**	1.261	**0.0090**	0.421	0.0628
I_{Rand}	300	**0 (opt)**	0.748	0.0090	0.738	0.0807
I_{Rand}	300	**0.0090**	1.383	2.6722	0.316	9.5768
I_{Rand}	700	**0 (opt)**	4.341	1.2751	3.670	4.4187
I_{Rand}	700	**0 (opt)**	10.490	0.0231	3.297	0.1387
I_{Rand}	700	**0 (opt)**	8.692	0.1694	3.246	0.6431
I_{Rand}	700	**0 (opt)**	8.298	0.1925	3.274	0.8163
I_{Rand}	700	**0 (opt)**	9.038	0.1155	3.204	0.4390

Table 7.1: Comparison of 2-approximation algorithms on conflicted instances

The results show that our algorithms find a better approximated solution in most of the instances. We even manage to find optimal solutions for most of the instances with

a *Rand* topology. The three algorithms are very fast and find solutions very close to an optimal solution.

Instance	$\|V\|$	Gap (%) Alg. 4+19	Time (s) Alg. 4+19	Gap (%) Alg. [82]	Time (s) Alg. [82]	Gap (%) Alg. 5
R_{Grid}	1000	0.0244	0.058	**0.0122**	0.021	0.0366
R_{Grid}	1000	**0.0122**	0.039	0.2455	0.014	0.3719
R_{Grid}	1000	0.0330	0.069	**0.0203**	0.027	0.1014
R_{Grid}	1000	**0.0048**	0.038	0.4006	0.016	0.6734
R_{Grid}	1000	**0.0888**	0.065	0.1037	0.020	0.1802
R_{Grid}	1500	**0.0083**	0.087	**0.0083**	0.045	0.0100
R_{Grid}	1500	**0.0100**	0.087	**0.0100**	0.042	0.0284
R_{Grid}	1500	**0.0807**	0.112	0.1066	0.042	0.1356
R_{Grid}	1500	**0.0344**	0.137	0.0410	0.033	0.0655
R_{Grid}	1500	**0.0287**	0.111	0.0439	0.051	0.0490
R_{Geom}	700	**0.0669**	0.068	0.2090	0.032	0.3845
R_{Geom}	700	**0.0755**	0.115	0.1283	0.033	0.1736
R_{Geom}	700	**0.0356**	0.063	0.0427	0.024	0.0783
R_{Geom}	700	**0.0651**	0.131	0.0732	0.030	0.1953
R_{Geom}	700	**0.0644**	0.093	**0.0644**	0.028	0.0967
R_{Geom}	1300	0.0128	0.529	**0.0043**	0.095	0.0427
R_{Geom}	1300	**0.0541**	0.321	0.1582	0.070	0.2540
R_{Geom}	1300	**0.1347**	0.202	0.1738	0.069	0.3216
R_{Geom}	1300	**0.0931**	0.504	**0.0931**	0.091	0.2031
R_{Geom}	1300	**0.1893**	0.190	**0.1893**	0.077	**0.1893**
R_{Rand}	300	**0 (opt)**	0.327	1.1364	0.384	6.8182
R_{Rand}	300	**0 (opt)**	0.308	**0 (opt)**	0.410	1.2500
R_{Rand}	300	**0 (opt)**	0.248	**0 (opt)**	0.426	**0 (opt)**
R_{Rand}	300	**0 (opt)**	0.221	**0 (opt)**	0.374	2.4691
R_{Rand}	300	**0 (opt)**	0.316	0.7407	0.412	2.2222
R_{Rand}	700	**0 (opt)**	2.046	**0 (opt)**	2.881	**0 (opt)**
R_{Rand}	700	**0 (opt)**	1.854	**0 (opt)**	2.806	**0 (opt)**
R_{Rand}	700	**0 (opt)**	2.049	**0 (opt)**	2.855	**0 (opt)**
R_{Rand}	700	**0 (opt)**	1.882	**0 (opt)**	2.477	**0 (opt)**
R_{Rand}	700	**0 (opt)**	2.288	**0 (opt)**	2.681	**0 (opt)**

Table 7.2: Comparison of 2-approximation algorithms on uncorrelated instances

7.3.2 Probing

The size of the problem has in general a large influence on the computational time. By exploiting the combinatorial structure of a minimum spanning tree $T^w = (V, F^w)$ for the edge weights, we efficiently detect *necessary* and *forbidden* edges, where a necessary

(respectively forbidden) edge is an edge that belongs to every (respectively none) feasible solution.

Proposition 34. *Let $e \in F^w$, if e is a bridge or if $\sum_{g \in F^w \cup \{f^*\} \setminus \{e\}} w_g > B$ where $f^* \in \arg\min_{f \in \delta(W) \setminus \{e\}} \{w_f\}$ where $W \subset V$ is the node set of one of the two created connected components when e is removed to T^w. Then e is a necessary edge.*

Proposition 35. *Let $e \in E \setminus F^w$, if $\sum_{g \in F^w \cup \{e\} \setminus \{f^*\}} w_g > B$ where $f^* \in \arg\max_{f \in C_e \setminus \{e\}} \{w_f\}$ where C_e is the cycle created by adding e to T^w. Then e is a forbidden edge.*

The proofs follow immediately from the cycle and cut optimality conditions. Algorithms 21 and 20 implement these probing rules.

Algorithm 20: Preprocessing 1 - Fixing necessary edges

Data: $G = (V, E)$, $w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, spanning tree $T^w = (V, F^w)$ minimizing $\sum_{e \in F^w} w_e$

$S \leftarrow \emptyset$;
for $e \in F^w$ **do**
  Let $f^* \in \arg\min_{f \in \delta(W) \setminus \{e\}} \{w_f\}$ where $W \subset V$ is the node set of one of the two created connected components when e is removed to T^w;
  if $\delta(W) = \{e\}$ or $w_{f^*} - w_e + \sum_{g \in F^w} w_g > B$ **then**
    $S \leftarrow S \cup \{e\}$;
  end
end
for $e \in S$ **do**
  Add the inequality $x_e = 1$ to the formulation;
end

Algorithm 21: Preprocessing 2 - Removing forbidden edges

Data: $G = (V, E)$, $w \in \mathbb{N}^{|E|}$, $B \in \mathbb{N}$, spanning tree $T^w = (V, F^w)$ minimizing $\sum_{e \in F^w} w_e$

$S \leftarrow \emptyset$;
for $e \in E \setminus F^w$ **do**
  Let $f^* \in \arg\max_{f \in C_e \setminus \{e\}} \{w_f\}$ where C_e is the cycle created by adding e to T^w;
  if $w_e - w_{f^*} + \sum_{g \in F^w} w_g > B$ **then**
    $S \leftarrow S \cup \{e\}$;
  end
end
Delete edges in S from E;

Algorithm 20 fixes a set of necessary edges. On the other hand, Algorithm 21 only keeps edges belonging to at least one feasible solution. Another preprocessing idea is based on an extension of a pruning rule proposed by Tarjan [74] for the minimum spanning tree to handle a budget constraint. Spanjaard and Sourd [72] used a similar procedure to remove dominated edges.

Proposition 36. *Let $T = (V, F)$ be a feasible solution for the 1BSTP. For any $e \in E \backslash F$, if every edge $f \in F$ in the cycle created by adding e to the graph T verifies $c_f \leq c_e$ and $w_f \leq w_e$, then there exists an optimal solution of the 1BSTP that does not contain e.*

The proof of this result is similar to the single objective case [74]. In practice, the MIP Start solution is a good choice for the spanning tree T in Proposition 36, Algorithm 22 implements this idea.

Algorithm 22: Preprocessing 3 - Removing dominated edges

```
Data: G = (V, E), T = (V, F) a feasible solution, w ∈ ℕ^|E|, c ∈ ℝ_+^|𝔼|
for e ∈ E\F do
    Let C_e be the cycle created by adding e to T;
    rem ← true;
    for f ∈ C_e\{e} do
        if w_e < w_f or c_e < c_f then
            rem ← false;
        end
    end
    if rem then
        Delete e from E;
    end
end
```

Table 7.3 reports the running time of our probing procedure and the average percentage of forbidden and necessary edges.

Instance	$\|V\|$	Edges removed or fixed by Alg. 20, 21 and 22 (%)	Time (s)	Instance	$\|V\|$	Edges removed or fixed by Alg. 20, 21 and 22 (%)	Time (s)
I_{Grid}	100	0	0.001	R_{Grid}	100	11.3	0.007
I_{Grid}	400	0	0.007	R_{Grid}	400	10.3	0.009
I_{Grid}	700	0	0.015	R_{Grid}	700	8.7	0.025
I_{Grid}	1000	0	0.040	R_{Grid}	1000	8.9	0.402
I_{Geom}	100	0	0.001	R_{Geom}	100	36.9	0.001
I_{Geom}	400	0.4	0.015	R_{Geom}	400	34.3	0.015
I_{Geom}	700	0.3	0.051	R_{Geom}	700	33.1	0.044
I_{Geom}	1000	0.3	0.098	R_{Geom}	1000	33.0	0.105
I_{Rand}	100	0.2	0.017	R_{Rand}	100	39.9	0.008
I_{Rand}	400	0.2	0.354	R_{Rand}	400	45.2	0.413
I_{Rand}	700	0.3	1.159	R_{Rand}	700	6.3	1.679
I_{Rand}	1000	0.3	3.668	R_{Rand}	1000	5.0	3.754

Table 7.3: Number of edges removed or fixed, and running time of the preprocessing procedure

By construction of the conflicted instances, only few variables are fixed to 1 or removed

by Algorithms 20, 21, 22. On the contrary, the preprocessing step is effective for uncorrelated instances, with up to 45 percent of edges removed or fixed. The total preprocessing running time is negligible, hence we keep this step in our final algorithm for both the conflicted and the uncorrelated cases.

7.3.3 Cut generation

Cut generation techniques attempt to restrict the feasible region of the linear program solved at the root of the Branch-and-Cut algorithm so that its solutions are closer to integers. In our context, the subtour formulation has an exponential number of constraints. In practice, it is prohibitive to generate all of them at the root. Instead, by exploiting the structure of a basic optimal solution for the 1BST relaxed problem and the geometry of the Lagrangian problem, we efficiently generate a number of these inequalities that have proved their worth in practice. Recall that an optimal solution is a convex combination of a feasible spanning tree $T^<$ and an infeasible spanning tree $T^>$. The latter has a small cost. A good rule of thumb is to generate a number of subtour constraints formed by adding edges to $T^<$. Let $T^< = (V, F^<)$ be the solution returned by Algorithm 19 and z^* be the z-coordinate of an optimal solution of the Lagrangian relaxation. The following algorithm generates some subtour constraints for the STP in the neighborhood of $T^<$.

Algorithm 23: Generation of subtour constraints

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{|E|}$, $z^* \in \mathbb{R}_+$, $T^< = (V, F^<)$
$M \leftarrow \max\{c_e + z^* w_e \mid e \in F^<\}$;
for $e \in E \backslash F^<$ **do**
  if $c_e + z^* w_e \leq M$ **then**
    Add the subtour constraint $\sum_{e \in E[W]} x_e \leq |W| - 1$, where $W \subset V$ contains all vertices of the created cycle by adding e to $T^<$;
  end
end

The characterization of a basic optimal solution could also be exploited to generate a lifted cover inequality. Observe that $T^> = (V, F^>)$ induces the cover inequality $\sum_{e \in F^>} x_e \leq |V| - 2$. In general the resulting cover inequality is rather weak but it can be lifted using Algorithm 14.

Algorithm 24: Cover inequality induced by $T^>$

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{R}^{|E|}$, $T^> = (V, F^>)$
Lift the inequality $\sum_{e \in F^>} x_e \leq |V| - 2$ using Algorithm 14 and add it to the model;

This argument could be generalized for any minimum spanning tree $T^z = (V, F^z)$ for the composite edge costs $c_e + z w_e$, where $0 \leq z < z^*$. The following procedure is more effective in practice.

The formulation of the problem could be further enhanced by adding valid inequalities obtained by projecting the extended exact formulation discussed in Chapter 6. Given

$r_E \in \mathbb{R}^{|E|}$, we showed how to construct a ray $(r_{card}, r_E, r_{y_0}, r^{h_1}, r^{h_1}_{y_0}, \ldots, , r^{h_{|Q^*|}}, r^{h_{|Q^*|}}_{y_0})$ of the projection cone $\mathcal{C}$ such that the induced valid inequality $r_E x \leq r_{y_0}$ defines a proper face of the convex hull of $\mathcal{H}$. The right-hand side r_{y_0} is computed by solving problem (6.10)-(6.14). In addition, if $r_E \in \{0,1\}^{|E|}$, then the tightest r_{y_0} can be efficiently computed by the Lagrangian relaxation technique, see Section 4.2.2. Algorithm 25 exploits again the Lagrangian function to generate a number of such inequalities.

Algorithm 25: Selection of a r^E and computation of r^{y_0}

Data: $G = (V, E)$, $w \in \mathbb{N}^{|E|}$, $c \in \mathbb{R}^{|E|}_+$, $z^* \in \mathbb{R}_+$, $j \in \mathbb{N}$
for $i = 1, \ldots, j-1$ **do**
  Compute the minimum spanning tree $T = (V, F)$ of cost $\sum_{e \in F} c_e + \frac{iz^* w_e}{j}$;
  $S \leftarrow \{e \in E \mid w_e = w_f, f \in F\}$;
  Compute $Z = \max\{\sum_{e \in S} x_e \mid x \in conv(\mathcal{H})\}$ using Algorithm 8;
  if $Z < |V| - 1$ **then**
    Add the inequality $\sum_{e \in S} x_e \leq Z$ to the formulation;
  end
end

Table 7.4 shows the efficiency of the MIP Start and of these valid inequalities added at the root of the branching algorithm. We compare Algorithm 17, the Branch-and-Cut based on the subtour formulation, with an enhanced version of Algorithm 17 including a MIP Start computed by Algorithm 4 and 19, and inequalities generated at the root node by Algorithms 23,24 and 25. The resolution time of a Branch-and-Cut algorithm based on subtour formulation has systematically benefited from their adding at the root node. The gap between an optimal solution of the root node subproblem x^* and an optimal integer solution x' values is also reduced by half for the I_{Grid} and I_{Geom} instances (computed by $\frac{\sum_{e \in E} c_e x'_e}{\sum_{e \in E} c_e x^*_e} - 1$).

Instance	$\|V\|$	Resolution time Algo. 17 (s)	Root gap Algo. 17 (%)	Resolution time enhanced Algo. 17 (s)	Root gap enhanced Algo. 17 (%)
I_{Grid}	1000	58.658	1.43	**22.637**	0.86
I_{Grid}	1000	135.695	1.41	**9.389**	0.66
I_{Grid}	1000	92.674	1.73	**19.528**	0.77
I_{Grid}	1000	113.167	2.04	**104.830**	0.91
I_{Grid}	1000	77.807	1.72	**14.458**	0.94
I_{Grid}	1500	1088.819	1.60	**72.964**	0.72
I_{Grid}	1500	1056.958	1.53	**307.105**	0.76
I_{Grid}	1500	847.664	1.31	**53.973**	0.76
I_{Grid}	1500	1521.135	1.31	**140.467**	0.66
I_{Grid}	1500	478.136	1.57	**69.313**	0.72
I_{Geom}	700	50.618	1.58	**18.312**	0.83
I_{Geom}	700	74.978	1.26	**19.864**	0.57
I_{Geom}	700	110.060	1.38	**33.624**	0.73
I_{Geom}	700	89.738	1.90	**39.827**	1.11
I_{Geom}	700	79.195	1.26	**27.321**	0.72
I_{Geom}	1300	705.517	1.40	**197.612**	0.77
I_{Geom}	1300	2133.710	1.39	**197.612**	0.76
I_{Geom}	1300	889.249	1.40	**641.977**	0.89
I_{Geom}	1300	1400.741	1.41	**288.309**	0.75
I_{Geom}	1300	**977.231**	1.51	1352.891	0.81
I_{Rand}	300	75.049	0.17	**4.584**	0.15
I_{Rand}	300	114.250	0.14	**1.051**	0.12
I_{Rand}	300	68.479	0.13	**1.608**	0.12
I_{Rand}	300	44.519	0.15	**1.596**	0.13
I_{Rand}	300	67.871	0.11	**22.857**	0.10
I_{Rand}	500	3289.617	0.11	**107.756**	0.10
I_{Rand}	500	3285.771	0.11	**20.482**	0.09
I_{Rand}	500	3600.000	0.05	**84.147**	0.05
I_{Rand}	500	3600.000	0.10	**157.316**	0.08
I_{Rand}	500	3600.000	0.09	**70.108**	0.09

Table 7.4: Comparison of the subtour ILP formulations adding a MIP Start and inequalities at the root

7.4 Cutting plane algorithm

Up to this point, a number of valid inequalities have been added at the root node. Despite reduced integrality gap, solving the resulting integer program is still time consuming. In order to reduce the size of the branching tree, we efficiently separate spanning tree's inequalities as discussed in Section 6.1 and inequalities of the KP as discussed in Section 6.2. Due to the arbitrary values of the costs and weights, it is very challenging to define classes of valid inequalities that separate any fractional solution. The extended formulation pro-

vides a systematic separation procedure. However, its implementation is prohibitive. We leave it as an open question to efficiency implement some of these ideas.

Algorithm 26 aggregates all algorithms separating spanning tree inequalities, where K is an upper bound on the number of valid inequalities generated by Algorithm 12 and 13, we set $K = 5|V|$.

Algorithm 26: Separation algorithm for spanning tree inequalities

Data: $G = (V, E)$, $x^* \in [0:1]^{|E|}, K \in \mathbb{N}$
Consider the subgraph $G^* = (V, E^*)$ where $E^* = \{e \in E \mid x^*(e) > 0\}$;
if x^* is integer **then**
  Run Algorithm 11(E, x^*, G^*);
else
  if the total number of added inequalities is lower than K **then**
    if G^* is not a connected graph **then**
      Run Algorithm 12(E, x^*, G^*);
    else
      Run Algorithm 13(E, x^*, G^*);
    end
  end
end

Table 7.5 shows the efficiency of our separation Algorithm 26. We compare two Branch-and-Cut algorithms beginning with the subproblem $\min\{\sum_{e\in E} c_e x_e \mid \sum_{e\in E} x_e = |V| - 1, \sum_{e\in E} w_e x_e \leq B, x_e \in [0;1]^{|E|}\}$, one with the subtour inequalities separation Algorithm 18 and the other with separation Algorithm 26. The branching algorithm with our separation procedure outperforms the subtour based algorithm, its resolution and total separation times are systematically smaller. Moreover, Algorithm 26 generates fewer valid inequalities, but they are more effective in practice.

Instance	$\|V\|$	Resolution time Alg. 18 (s)	Separation time Alg. 18 (s)	Total number of separated inequalities Alg. 18	Resolution time Alg. 26 (s)	Separation time Alg. 26 (s)	Total number of separated inequalities Alg. 26
I_{Grid}	1000	79.167	29.15	12366	**6.841**	2.65	2224
I_{Grid}	1000	56.571	22.62	11561	**4.150**	1.28	2587
I_{Grid}	1000	86.393	33.85	14155	**1.469**	0.55	1718
I_{Grid}	1000	115.593	33.87	14098	**5.856**	1.77	2752
I_{Grid}	1000	94.631	37.40	16203	**1.043**	0.22	1475
I_{Grid}	1500	456.321	116.41	30777	**75.332**	13.28	5607
I_{Grid}	1500	833.485	204.91	37822	**17.386**	4.69	4369
I_{Grid}	1500	543.075	131.23	28543	**6.297**	1.72	3434
I_{Grid}	1500	494.202	136.97	31119	**3.064**	0.75	2583
I_{Grid}	1500	503.599	164.03	29638	**11.322**	3.10	3738
I_{Geom}	700	113.678	26.49	12575	**3.688**	0.92	2158
I_{Geom}	700	28.976	8.05	6959	**1.932**	0.40	2009
I_{Geom}	700	94.812	21.80	12671	**5.817**	1.26	2777
I_{Geom}	700	79.972	18.21	10733	**4.257**	0.87	2332
I_{Geom}	700	94.853	22.05	12681	**3.433**	0.74	2157
I_{Geom}	1300	1241.504	147.29	29397	**34.816**	5.68	5464
I_{Geom}	1300	543.402	75.40	23824	**13.749**	3.21	4089
I_{Geom}	1300	1155.248	157.51	39875	**54.789**	7.92	4889
I_{Geom}	1300	646.227	119.97	32960	**105.899**	12.82	7185
I_{Geom}	1300	1988.522	224.37	44153	**44.385**	7.03	5283
I_{Rand}	300	79.281	19.09	4007	**3.668**	2.07	572
I_{Rand}	300	79.342	19.53	3960	**49.965**	7.61	2493
I_{Rand}	300	108.968	22.89	3914	**37.976**	6.99	3128
I_{Rand}	300	**54.472**	18.81	3557	63.582	9.06	3344
I_{Rand}	300	257.568	35.56	6236	**18.604**	6.27	1632
I_{Rand}	700	3600.000	500.49	8395	3600.000	214.40	16387
I_{Rand}	700	3600.000	467.95	7532	**425.909**	79.38	638
I_{Rand}	700	3600.000	510.31	5663	**1626.738**	127.97	10018
I_{Rand}	700	3600.000	578.15	6899	**749.095**	80.78	5565
I_{Rand}	700	3600.000	505.71	8175	3600.000	1.17	13042

Table 7.5: Comparison of separation algorithms 18 and 26

The following Algorithm 27 aggregates the procedures separating inequalities combining the spanning tree and the knapsack parts. We set $K' = 0.5|V|$.

Algorithm 27: Separation algorithm for cover inequalities

Data: $G = (V, E)$, $x^* \in [0:1]^{|E|}, w \in \mathbb{N}^{|E|}, c \in \mathbb{R}_+^{|E|}, B, K' \in \mathbb{N}$
Consider the subgraph $G^* = (V, E^*)$ where $E^* = \{e \in E \mid x^*(e) > 0\}$;
if x^* is fractional and G^* is connected **then**
  Let $T^* = (V, F^*)$ be a spanning tree maximizing $\sum_{e \in F^*} x_e^*$;
  if $\sum_{e \in F^*} w_e > B$ and $\sum_{e \in F^*} x_e^* > |V| - 2$ **then**
    Lift the inequality $\sum_{e \in F^*} x_e \leq |V| - 2$ using Algorithm 14 and add it to the model;
  end
  if the total number of added inequalities is lower than K' **then**
    Run Algorithm 15(G, w, c, B, x^*);
  end
end

7.5 Algorithm for the minimum 1-budgeted spanning tree problem

Until this point, in this chapter we introduced several procedures that separately improve the resolution time of the M1BSTP. In this Section, we present an algorithm combining all of those procedures such that it will benefice from all improvements.

Algorithm 28: Optimization procedure for M1BSTP

Data: $G = (V, E), w \in \mathbb{N}^{|E|}, c \in \mathbb{R}_+^{|E|}, B \in \mathbb{N}$

1 Compute a spanning tree $T^w = (V, F^w)$ minimizing $\sum_{e \in F^w} w_e$;
2 **if** $\sum_{e \in F^w} w_e > B$ **then**
3 | Stop, there is no feasible solution;
4 **end**
5 # **Calling preprocessing procedures**
6 Run Algorithm 20 (G, w, B, T^w);
7 Run Algorithm 21 (G, w, B, T^w);
8 # **Compute the Lagrangian relaxation**
9 Run Algorithm 4, let $T^<$ (respectively $T^>$) be the feasible (respectively infeasible) spanning tree obtained by rounding the Lagrangian relaxation and z^* the coordinate of the maximum;
10 # **Ending preprocessing procedure**
11 $T^< \leftarrow$ Algorithm 19 $(G, c, w, B, T^<)$;
12 Run Algorithm 22 $(G, T^<, w, c)$;
13 Build the Integer Linear Problem $\min\{\sum_{e \in E} c_e x_e \mid x(E) = |V| - 1, \sum_{e \in E} w_e x_e \leq B, x \in \mathbb{B}^{|E|}\}$;
14 Set $T^<$ as a MIP Start;
15 # **Add subtour constraints to the root**
16 Run Algorithm 23 $(G, c, w, z^*, T^<)$;
17 # **Add valid cover inequality**
18 Run Algorithm 24 $(G, c, w, T^>)$;
19 #**Add valid inequalities via the projection technique**
20 Run Algorithm 25 $(G, w, c, z^*, 6)$;
21 Solve the ILP by Branch-and-Cut, with the separation algorithm for subtour inequalities 26 and the separation algorithm for cover inequalities 27 ;

We summarize in Algorithm 28 the essential elements of the Branch-and-Cut algorithm. It begins with the preprocessing and probing procedures (Section 7.3.2) intended to reduce the size of the formulation (lines 5-7 and 10-12). The algorithm also solves the Lagrangian relaxation in advance of the start of the procedure in order to compute a MIP Start solution (lines 8-9). This step is valuable to quickly allows the algorithm to get close to optimality. Furthermore, the geometry of the Lagrangian function is exploited to efficiently generate valid inequalities related to the spanning tree and the knapsack polytope (lines 15-18). The formulation of the problem is further strengthened by adding valid inequalities obtained via the projection technique (lines 19-20). After the presolve step, the Branch-and-Cut algorithm continues to strengthen the LP relaxation during the search process. This is accomplished by calling the separation algorithms 26 and 27 to generate cutting planes in the nodes of the Branch-and-Bound tree. The efficiency of Algorithm 28 is illustrated in the next section, comparing its resolution time with other formulations.

7.6 Computational experiments

The results of the experiments are presented in the following Tables. In a search for comparison, we used several metrics:

- RT_X: total resolution time of strategy X, given in seconds,
- NN_X: total number of nodes in the branching tree of strategy X,
- Gap_X: gap in percent between the best integer solution found and the best lower bound at the end of the resolution of strategy X. In the case where no feasible solution has been found, we write $+\infty$.

Where strategy X stands for:

- ST: Subtour formulation, Algorithm 17,
- $Alg28$: Algorithm 28, and
- MTZ: MTZ formulation (3.14)-(3.20) with inequalities (7.1).

First, we consider conflicted instances. Then, we consider uncorrelated instances. Finally, we give additional statistics on separation algorithms and study variants of Algorithm 28.

7.6.1 Conflicted instances

Table 7.6 reports the performances of the three algorithms on instances I_{Grid}.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg28}	NN_{28}	Gap_{Alg28}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	0.535	7	0	**0.035**	12	0	29.051	166842	0
100	0.074	1	0	**0.037**	33	0	0.228	972	0
100	0.132	80	0	**0.058**	63	0	0.507	1318	0
100	0.166	129	0	**0.050**	41	0	0.533	3413	0
100	0.154	87	0	**0.036**	37	0	0.805	5831	0
500	3.332	291	0	**0.471**	156	0	51.897	46261	0
500	3.361	315	0	**0.312**	124	0	3600.000	3280040	0.0178
500	10.109	339	0	**0.499**	227	0	78.907	98461	0
500	2.968	311	0	**0.391**	119	0	89.672	128648	0
500	3.101	347	0	**0.346**	103	0	274.296	562978	0
1000	98.478	598	0	**3.527**	755	0	3600.000	3793796	0.0729
1000	164.094	583	0	**2.996**	336	0	1121.181	633963	0
1000	67.403	425	0	**0.571**	125	0	2866.763	1721204	0
1000	59.941	479	0	**0.641**	122	0	3600.000	2398244	0.0155
1000	130.483	554	0	**5.863**	1009	0	3600.000	3051671	$+\infty$
1500	430.087	734	0	**6.274**	3609	0	3600.000	1572891	0.1377
1500	845.311	740	0	**25.469**	2129	0	3600.000	1496482	0.1057
1500	590.696	884	0	**70.491**	4835	0	3600.000	1660841	$+\infty$
1500	402.476	830	0	**128.777**	7117	0	3600.000	1732946	$+\infty$
1500	352.690	703	0	**33.872**	2667	0	3600.000	1856560	$+\infty$
2000	2845.486	1083	0	**21.105**	1625	0	3600.000	788043	$+\infty$
2000	3600.000	1916	$+\infty$	**1234.588**	17801	0	3600.000	864020	$+\infty$
2000	3600.000	1618	$+\infty$	**670.009**	10376	0	3600.000	721558	$+\infty$
2000	2387.236	1031	0	**62.544**	3081	0	3600.000	745501	$+\infty$
2000	3600.000	1045	0.0089	**13.888**	928	0	3600.000	786008	$+\infty$
2500	3600.000	1020	0.3221	**136.787**	3319	0	3600.000	498091	$+\infty$
2500	3600.000	1545	0.3608	**3600.000**	35182	0.0062	3600.000	485440	$+\infty$
2500	3600.000	4296	$+\infty$	**67.084**	2868	0	3600.000	521056	$+\infty$
2500	3600.000	4101	$+\infty$	**3578.085**	50561	0	3600.000	464820	$+\infty$
2500	3600.000	3709	$+\infty$	**484.597**	9517	0	3600.000	455711	$+\infty$
3000	3600.000	969	$+\infty$	**3600.000**	22746	0.0162	3600.000	375345	$+\infty$
3000	3600.000	4822	$+\infty$	**3600.000**	22951	0.0015	3600.000	359462	$+\infty$
3000	3600.000	1106	$+\infty$	**1214.479**	12258	0	3600.000	401046	$+\infty$
3000	3600.000	1045	$+\infty$	**1319.695**	13494	0	3600.000	384423	$+\infty$
3000	3600.000	4025	$+\infty$	**3600.000**	20765	0.0125	3600.000	376842	$+\infty$

Table 7.6: Instances I_{Grid}

We observe that our algorithm outperforms the subtour and MTZ formulations. The latter strategy is the worst for this class of problem and times out after one hour (without even finding a feasible solution) for moderate instances of even 1000 nodes. The subtour formulation has a better performance and times out after one hour (without even finding a feasible solution), when solving large instances with even 2500 nodes. Although Algorithm 28 times out for instances with 3000 nodes, the final gap is very small ($< 0.02\%$). Using a MIP Start in Algorithm 28 guarantees to find a feasible solution for higher instance sizes.

Table 7.7 shows that Algorithm 28 also dominates the ST and MTZ strategies on instances I_{Geom}. Similarly to instances I_{Grid}, the MTZ formulation performs poorly and times out after one hour even for moderate sized graphs with 500 nodes. For large instances with 2000 nodes, all algorithms time out after one hour. In contrast with ST and MTZ strategies (which were not able to find a feasible solution), our algorithm succeed to solve two instances, otherwise it has a very small final gap ($< 0.0072\%$).

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg28}	NN_{28}	Gap_{Alg28}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	0.186	59	0	**0.040**	59	0	2.679	15523	0
100	0.153	28	0	**0.081**	47	0	0.129	1	0
100	0.379	59	0	**0.112**	1	0	1.701	3901	0
100	0.345	87	0	**0.163**	53	0	1.190	1235	0
100	0.199	7	0	**0.160**	1	0	34.302	120634	0
500	13.339	352	0	**0.766**	190	0	3081.545	1958908	0
500	18.623	505	0	**4.650**	517	0	3600.000	2776632	0.0289
500	11.376	358	0	**4.025**	634	0	3600.000	4958635	0.0629
500	17.590	429	0	**1.859**	323	0	3600.000	2489120	0.1017
500	22.512	467	0	**11.964**	1788	0	3600.000	2984276	0.3307
1000	334.600	517	0	**103.484**	3890	0	3600.000	1468427	0.1012
1000	161.114	809	0	**33.893**	2180	0	3600.000	1794150	$+\infty$
1000	273.269	905	0	**135.620**	6525	0	3600.000	1534070	0.1757
1000	487.522	735	0	**155.229**	6857	0	3600.000	1725068	0.0995
1000	376.945	696	0	**12.477**	1133	0	3600.000	1456834	$+\infty$
1500	2816.989	1027	0	**471.637**	31005	0	3600.000	726849	$+\infty$
1500	**1781.083**	1141	0	1952.863	68101	0	3600.000	813840	$+\infty$
1500	2513.186	1475	0	**372.601**	7494	0	3600.000	801773	$+\infty$
1500	1716.585	1027	0	**289.540**	4465	0	3600.000	763420	$+\infty$
1500	3600.000	1027	0.0016	**336.866**	5414	0	3600.000	771774	$+\infty$
2000	3600.000	2266	$+\infty$	**3600.000**	52392	0.0060	3600.000	314741	$+\infty$
2000	3600.000	5232	$+\infty$	**2576.873**	40340	0	3600.000	342338	$+\infty$
2000	3600.000	1301	$+\infty$	**3600.000**	48634	0.0024	3600.000	386501	$+\infty$
2000	3600.000	3887	$+\infty$	**3600.000**	58295	0.0072	3600.000	375011	$+\infty$
2000	3600.000	4337	$+\infty$	**1571.902**	18433	0	3600.000	312730	$+\infty$

Table 7.7: Instances I_{Geom}

Table 7.8 does not show a clear dominance in terms of computational time of the MTZ or Algorithm 28 strategies in moderate sized I_{Rand} instances.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg28}	NN_{28}	Gap_{Alg28}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	0.473	19	0	0.466	1	0	**0.347**	1	0
100	0.604	23	0	0.506	1	0	**0.212**	1	0
100	0.544	31	0	**0.460**	1	0	6.736	46435	0
100	0.495	87	0	**0.259**	1	0	0.328	1	0
100	0.241	59	0	**0.074**	1	0	379.216	210195	0
500	3600.000	2831	0.0054	**9.017**	1	0	9.264	1	0
500	3600.000	3832	0.0054	**9.371**	1	0	162.161	317049	0
500	3600.000	2319	0.0593	104.255	56296	0	**32.915**	1123	0
500	3600.000	3086	0.0477	**10.805**	1	0	15.410	13	0
500	3600.000	2638	0.0513	**9.528**	1	0	11.961	1	0
1000	3600.000	1422	$+\infty$	**61.312**	50	0	90.689	1	0
1000	3600.000	1673	$+\infty$	160.498	4242	0	**91.285**	1	0
1000	3600.000	1589	$+\infty$	244.048	1066	0	**75.578**	1	0
1000	3600.000	1660	$+\infty$	**126.259**	282	0	242.288	1	0
1000	3600.000	1709	$+\infty$	119.720	206	0	**113.155**	1	0
1500	3600.000	730	$+\infty$	**136.748**	1	0	343.828	1	0
1500	3600.000	695	$+\infty$	**97.252**	1	0	354.614	35	0
1500	3600.000	801	$+\infty$	**120.466**	1	0	288.068	1	0
1500	3600.000	734	$+\infty$	3600.000	880874	0.0018	**256.551**	1	0
1500	3600.000	680	$+\infty$	**156.070**	9	0	293.806	1	0
2000	3600.000	599	$+\infty$	**283.43**	1	0	1501.518	413	0
2000	3600.000	578	$+\infty$	**191.531**	1	0	3600.000	2516	0.0032
2000	3600.000	612	$+\infty$	3600.000	337741	0.0014	**593.772**	1	0
2000	3600.000	575	$+\infty$	**201.176**	1	0	456.621	1	0
2000	3600.000	580	$+\infty$	**231.799**	1	0	680.354	1	0
2500	3600.000	523	$+\infty$	**372.427**	1	0	845.551	1	0
2500	3600.000	404	$+\infty$	**372.504**	1	0	818.083	1	0
2500	3600.000	420	$+\infty$	**473.371**	1	0	1598.928	1	0
2500	3600.000	607	$+\infty$	**3600.000**	28647	0.0011	3600.000	318	0.0032
2500	3600.000	431	$+\infty$	**467.149**	1	0	781.916	1	0
3000	3600.000	181	$+\infty$	**657.939**	1	0	2291.782	2638	0
3000	3600.000	179	$+\infty$	**663.416**	1	0	938.007	1	0
3000	3600.000	206	$+\infty$	**443.146**	1	0	873.496	1	0
3000	3600.000	210	$+\infty$	3600.000	459	0.0072	**1944.794**	1	0
3000	3600.000	184	$+\infty$	**594.435**	1	0	3600.000	4348	0.0009

Table 7.8: Instances I_{Rand}

The performance of both Algorithm 28 and the MTZ are highly volatile. They solve most of the instances at the root node, otherwise they may be timed out after one hour. Algorithm 28 benefits from the MIP Start which is very close to an optimal solution (see Table 7.1), in this case our algorithm is the fastest. For the *Rand* topology, solvers exploit the MTZ formulation with a set of preprocessing routines allowing one to solve quickly the problem. Note that if the latter are disabled, the resulting formulation would have the highest resolution time. The Table also demonstrates that the subtour formulation

encounters difficulty in solving even small sized instances contrarily to the previous instance classes.

7.6.2 Uncorrelated instances

Table 7.9 shows that the three algorithms solve instances R_{Grid} of the same size as conflicted instances (see Table 7.6).

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg28}	NN_{28}	Gap_{Alg28}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	0.015	1	0	**0.012**	10	0	0.041	1	0
100	0.675	47	0	**0.069**	7	0	0.339	1268	0
100	0.070	1	0	**0.026**	1	0	0.325	2514	0
100	0.055	1	0	**0.050**	30	0	0.417	2514	0
100	0.090	8	0	**0.040**	1	0	0.368	3202	0
500	8.180	428	0	**0.783**	396	0	215.328	431151	0
500	6.738	480	0	**0.743**	323	0	140.803	213540	0
500	2.512	321	0	**0.353**	151	0	25.557	34685	0
500	4.882	278	0	**0.554**	329	0	29.707	52319	0
500	8.014	355	0	**0.454**	101	0	17.385	6661	0
1000	105.147	495	0	**3.794**	665	0	583.193	494067	0
1000	54.596	735	0	**3.912**	878	0	3600.000	2174463	0.0790
1000	37.540	838	0	**1.401**	386	0	3600.000	2499835	0.0289
1000	157.389	527	0	**2.867**	575	0	3600.000	2679277	0.2470
1000	97.608	457	0	**2.159**	424	0	3600.000	2689377	0.1196
1500	667.951	869	0	**2.328**	246	0	3600.000	1704460	$+\infty$
1500	195.898	594	0	**2.177**	328	0	3600.000	1658874	$+\infty$
1500	847.460	1017	0	**122.261**	6551	0	3600.000	1419249	$+\infty$
1500	448.779	788	0	**22.271**	1946	0	3600.000	1598070	0.7017
1500	674.585	878	0	**30.644**	2735	0	3600.000	1504278	$+\infty$
2000	1461.395	1042	0	**186.058**	5666	0	3600.000	897177	$+\infty$
2000	1056.126	1312	0	**71.710**	4030	0	3600.000	2823155	0.5496
2000	3600.000	1060	0.7995	**79.635**	26965	0	3600.000	926155	$+\infty$
2000	3600.000	2400	$+\infty$	**975.322**	22430	0	3600.000	1145988	$+\infty$
2000	1827.709	963	0	**42.820**	2725	0	3600.000	941024	$+\infty$
3000	3600.000	799	$+\infty$	**1593.351**	27939	0	3600.000	1048253	$+\infty$
3000	3600.000	769	$+\infty$	**3600.000**	32623	0.0032	3600.000	946081	$+\infty$
3000	3600.000	4810	$+\infty$	**3600.000**	22498	0.0309	3600.000	371146	$+\infty$
3000	3600.000	756	$+\infty$	**1033.676**	11246	0	3600.000	453082	$+\infty$
3000	3600.000	3902	$+\infty$	**3600.000**	27111	0.0151	3600.000	315088	$+\infty$
4000	3600.000	5228	$+\infty$	**3600.000**	16934	0.0352	3600.000	277769	$+\infty$
4000	3600.000	940	$+\infty$	**3600.000**	18608	0.0096	3600.000	301842	$+\infty$
4000	3600.000	4221	$+\infty$	**3600.000**	20643	0.0194	3600.000	289433	$+\infty$
4000	3600.000	825	$+\infty$	**3600.000**	17836	0.0346	3600.000	341776	$+\infty$
4000	3600.000	5108	$+\infty$	**3600.000**	17076	0.0659	3600.000	280636	$+\infty$

Table 7.9: Instances R_{Grid}

The subtour formulation is systematically timed out (without even find a feasible solution) on large instances with 3000 nodes or more and the MTZ formulation is systematically timed out (without even find a feasible solution) on moderate instances with 1000 nodes or more. Our algorithm outperforms the subtour and MTZ strategies.

Table 7.10 shows that the three algorithms succeed to solve instances of the same size as conflicted instances (see Table 7.7). Algorithm 28 remains the best choice for instances R_{Geom}.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg28}	NN_{28}	Gap_{Alg28}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	0.217	185	0	**0.044**	31	0	6.669	40480	0
100	0.195	183	0	**0.099**	90	0	3600.000	35589899	1.0782
100	0.162	79	0	**0.065**	65	0	42.358	202084	0
100	0.161	107	0	**0.046**	22	0	3600.000	7248760	2.3602
100	0.242	271	0	**0.074**	61	0	5.112	43515	0
500	21.295	483	0	**2.689**	574	0	203.484	343445	0
500	8.659	506	0	**1.199**	377	0	3600.000	4083926	0.6395
500	**10.304**	384	0	11.711	812	0	3600.000	3651555	0.4180
500	7.851	353	0	**1.607**	450	0	3600.000	4410912	0.8912
500	12.604	406	0	**1.082**	287	0	3600.000	3531740	0.2341
1000	492.933	818	0	**112.253**	6564	0	3600.000	1855247	0.7759
1000	**198.022**	761	0	1483.203	135585	0	3600.000	1768854	0.7228
1000	**259.415**	604	0	345.594	21689	0	3600.000	1671701	0.2580
1000	232.163	697	0	**10.818**	947	0	3600.000	1722629	0.6541
1000	**184.685**	698	0	256.864	8353	0	3600.000	2091388	1.2306
1500	2848.403	1271	0	**238.887**	7420	0	3600.000	844034	$+\infty$
1500	3306.238	1088	0	**616.578**	20976	0	3600.000	699890	$+\infty$
1500	3600.000	998	1.4816	**1152.348**	23492	0	3600.000	745518	$+\infty$
1500	3600.000	1053	4.1198	**3600.000**	103740	0.1643	3600.000	779040	$+\infty$
1500	3600.000	1020	$+\infty$	**2451.702**	79003	0	3600.000	812749	$+\infty$
2000	3600.000	2200	$+\infty$	**3600.000**	154427	0.0794	3600.000	413602	$+\infty$
2000	3600.000	919	$+\infty$	**3600.000**	64325	0.1337	3600.000	400798	$+\infty$
2000	3600.000	1189	$+\infty$	**3600.000**	41268	0.1323	3600.000	638739	$+\infty$
2000	3600.000	1205	$+\infty$	**3600.000**	37827	0.0225	3600.000	500189	$+\infty$
2000	3600.000	1403	$+\infty$	**3600.000**	50107	0.1450	3600.000	487006	$+\infty$

Table 7.10: Instances R_{Geom}

Table 7.11 illustrates the efficiency of our algorithm in instances R_{Rand}. The combination of the high density of the graph and the random generation of uncorrelated costs and weights values favors the existence of a large numbers of feasible solutions with a very good quality. Such solutions can be found quickly running Algorithms 4 and 19. The latter solution, used as a MIP Start, allows one to solve problems at the root of the Branch-and-Cut algorithm. In contrast with the conflicted instances, the MTZ strategy is slower than our algorithm in uncorrelated instance.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg28}	NN_{28}	Gap_{Alg28}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	0.164	59	0	0.132	1	0	**0.087**	1	0
100	0.159	23	0	**0.085**	1	0	0.152	1	0
100	0.212	31	0	**0.119**	1	0	2.158	8163	0
100	0.169	39	0	**0.142**	7	0	7.256	103916	0
100	0.452	71	0	**0.104**	1	0	0.781	1470	0
500	3.776	57	0	**1.859**	1	0	3.576	1	0
500	4.405	36	0	**1.170**	1	0	2.066	1	0
500	**1.095**	1	0	2.007	1	0	3.317	1	0
500	5.135	36	0	**2.284**	1	0	3.467	1	0
500	**1.450**	1	0	2.608	1	0	3.452	1	0
1000	3600.000	1733	$+\infty$	**16.579**	1	0	42.008	1	0
1000	3600.000	1907	$+\infty$	**13.348**	1	0	36.779	1	0
1000	3600.000	2105	$+\infty$	**29.403**	7	0	46.283	1	0
1000	3600.000	1907	$+\infty$	**2.896**	1	0	64.314	1	0
1000	3600.000	1887	$+\infty$	**5.146**	1	0	40.898	1	0
2000	3600.000	669	$+\infty$	**20.099**	1	0	211.766	1	0
2000	3600.000	622	$+\infty$	**19.699**	1	0	201.575	1	0
2000	3600.000	702	$+\infty$	**18.721**	1	0	195.629	1	0
2000	3600.000	685	$+\infty$	**21.746**	1	0	257.610	1	0
2000	3600.000	690	$+\infty$	**20.689**	1	0	359.472	1	0
3000	3600.000	216	$+\infty$	**256.258**	1	0	1279.278	1	0
3000	3600.000	210	$+\infty$	**207.036**	1	0	1317.434	1	0
3000	3600.000	222	$+\infty$	**196.612**	1	0	916.191	1	0
3000	3600.000	198	$+\infty$	**45.706**	1	0	1085.403	1	0
3000	3600.000	223	$+\infty$	**57.808**	1	0	891.754	1	0

Table 7.11: Instances R_{Rand}

7.6.3 Separation statistics and variants

In Table 7.12, we compare the three strategies with metrics related to the separation phase:

- ST_X: Total separation time of strategy X, given in seconds,
- NC_X: Total number of valid inequalities separated of strategy X,
- NN_X: Total number of nodes in the branching tree of strategy X.

Table 7.12 gives the average values of the above metrics. Note that for Algorithm 28, we divide the separated inequalities in two categories; $NC(ST)$ related to spanning tree inequalities, separated by Algorithm 26, and $NC(C)$ associated with cover inequalities separated by Algorithm 27.

Instances	$\|V\|$	ST_{ST}	NC_{ST}	NN_{ST}	ST_{Alg28}	$NC_{Alg28}(ST)$	$NC_{Alg28}(C)$	NN_{Alg28}	NN_{MTZ}
I_{Grid}	100	0.03	117	15	0.01	195	15	58	1150
I_{Grid}	500	3.38	3343	325	0.25	746	7	146	823278
I_{Grid}	1000	45.30	16215	528	2.07	1547	52	469	2319776
I_{Grid}	1500	137.71	26706	778	19.59	4137	48	4071	1663944
I_{Grid}	2000	564.45	66798	1339	33.01	5684	38	7092	781026
I_{Grid}	2500	788.36	82113	2934	60.70	8991	49	15525	485024
I_{Grid}	3000	1097.04	83695	2393	131.20	14744	95	18443	379424
I_{Geom}	100	0.22	349	147	0.01	164	2	38	10084
I_{Geom}	500	6.88	4641	422	1.82	1472	59	690	2691009
I_{Geom}	1000	112.91	29509	732	11.84	3706	51	2766	1156951
I_{Geom}	1500	275.17	52789	1056	31.44	7285	129	23296	775531
I_{Geom}	2000	667.58	81914	3956	51.57	100009	37	35705	346264
I_{Rand}	100	0.23	212	79	0.02	68	1	1	1
I_{Rand}	500	430.10	18454	2941	4.57	841	0	11260	63411
I_{Rand}	1000	634.42	14985	1611	82.30	3536	0	1169	1
I_{Rand}	1500	689.89	11845	698	110.20	2693	0	176177	8
I_{Rand}	2000	1020.30	9182	602	112.30	2694	0	67549	586
I_{Rand}	2500	1522.77	8904	515	216.08	3064	0	57296	64
I_{Rand}	3000	1802.39	8042	437	475.78	3270	0	93	1398

Table 7.12: Comparison of separation times, number of separated inequalities and number of nodes for the three algorithms

Table 7.12 shows that the subtour formulation has systematically more time spent in separation phases and more separated inequalities. In practice, all those added inequalities slow the computations of the subproblems, and therefore few nodes are explored compared to Algorithm 28 and MTZ. The results show that most of the separated inequalities in Algorithm 28 are spanning tree inequalities. Although the cover based inequalities are very effective, our spanning tree interdiction heuristics generate valid inequalities only in few cases, especially in the *Rand* topology. Table 7.12 also shows that the number of nodes in the branching tree of the MTZ formulation explodes in instances I_{Grid} and I_{Geom} compared to the other strategies.

The following Tables compare our Algorithm 28 with two other variants:

- $Var1$: Algorithm 28 without MIP Start (remove line 14),
- $Var2$: Algorithm 28 without MIP Start, constraints added at the root and separation algorithm 27 (remove lines 14-20 and modify line 21, it only remains preprocessing and our spanning tree's inequalities separation algorithm).

Tables 7.13 and 7.14 give the results in instances I_{Grid} and R_{Grid}. They do not show a clear dominance among the three algorithms. The variant $Var1$ appears to be more efficient in average for instances with 3000 nodes or more.

$\|V\|$	RT_{Algo28}	NN_{Algo28}	Gap_{Algo28}	RT_{Var1}	NN_{Var1}	Gap_{Var1}	RT_{Var2}	NN_{Var2}	Gap_{Var2}
500	0.374	130	0	**0.302**	80	0	0.309	151	0
500	**0.113**	61	0	0.154	58	0	0.377	178	0
500	0.164	64	0	**0.121**	58	0	0.454	241	0
500	0.314	189	0	**0.142**	58	0	0.390	263	0
500	**0.173**	75	0	0.200	97	0	0.612	299	0
1000	5.211	985	0	**2.511**	621	0	4.314	1040	0
1000	2.565	662	0	**1.646**	499	0	1.978	454	0
1000	2.671	579	0	4.077	965	0	**2.297**	616	0
1000	1.710	396	0	**1.620**	551	0	1.731	475	0
1000	**2.337**	728	0	3.041	722	0	3.579	779	0
1500	**6.597**	916	0	9.704	1163	0	6.978	811	0
1500	**12.873**	1502	0	19.802	1808	0	13.804	1383	0
1500	105.881	5114	0	**10.090**	1088	0	248.243	6734	0
1500	23.340	2128	0	95.830	4914	0	**10.270**	962	0
1500	175.401	7462	0	**2.708**	1847	0	107.225	5405	0
2000	**519.893**	8962	0	1222.620	20301	0	1910.764	54524	0
2000	**17.103**	1913	0	17.594	1641	0	55.694	2956	0
2000	838.263	10696	0	874.137	20876	0	**58.523**	2453	0
2000	**35.790**	3104	0	61.554	3553	0	56.236	1880	0
2000	305.996	7246	0	**54.155**	2529	0	62.976	2808	0
2500	3600.000	56386	0.0018	162.093	5237	0	**150.781**	28839	0
2500	1976.952	20602	0	**752.600**	11202	0	854.005	12093	0
2500	848.978	12253	0	689.652	13169	0	**278.648**	6488	0
2500	1091.845	15948	0	420.771	7834	0	**290.588**	5643	0
2500	905.459	10323	0	**88.178**	3074	0	98.921	2604	0
3000	306.464	6837	0	**184.814**	4752	0	706.357	6532	0
3000	3600.000	18256	0.0030	**574.872**	27431	0	3600.000	33042	0.0045
3000	249.896	4773	0	213.572	4980	0	**154.102**	3598	0
3000	3600.000	25527	0.0050	**1943.115**	2370	0	3600.000	32511	0.002
3000	3600.000	26671	0.0030	420.658	6540	0	**200.154**	4935	0
3500	3600.000	19783	0.0070	3600.000	18274	0.0057	**2596.669**	15499	0
3500	3600.000	29750	0.0038	**3600.000**	32154	0.0032	**3600.000**	26233	0.0032
3500	3200.360	23608	0	3600.000	27108	0.0032	**2932.924**	35124	0
3500	1754.113	14705	0	**222.746**	4316	0	242.003	4662	0
3500	**243.001**	3764	0	403.789	6336	0	253.903	4443	0

Table 7.13: Comparison among Algorithm 28, $Var1$ and $Var2$ on instances I_{Grid}

$\|V\|$	RT_{Algo28}	NN_{Algo28}	Gap_{Algo28}	RT_{Var1}	NN_{Var1}	Gap_{Var1}	RT_{Var2}	NN_{Var2}	Gap_{Var2}
500	**0.353**	228	0	0.366	196	0	0.398	283	0
500	**0.118**	64	0	0.188	103	0	0.366	197	0
500	0.278	142	0	**0.182**	87	0	0.261	175	0
500	**0.275**	177	0	0.289	152	0	0.612	421	0
500	0.631	368	0	0.448	227	0	**0.263**	124	0
1000	**0.466**	100	0	1.149	350	0	2.085	602	0
1000	4.279	970	0	**2.969**	827	0	4.332	1336	0
1000	**0.669**	187	0	0.779	221	0	1.803	570	0
1000	4.710	927	0	4.091	830	0	**3.242**	752	0
1000	2.216	41	0	**0.452**	133	0	1.919	490	0
2000	1723.998	29095	0	**66.191**	2794	0	1003.092	16751	0
2000	629.789	16422	0	**464.902**	13632	0	1202.387	9503	0
2000	92.899	4274	0	**49.548**	2863	0	133.813	4425	0
2000	39.359	2524	0	**6.262**	632	0	11.794	952	0
2000	713.278	13038	0	505.745	12775	0	**469.689**	8772	0
3000	715.498	10180	0	1236.226	11474	0	**379.247**	5912	0
3000	1550.958	20833	0	999.211	7667	0	**358.780**	5889	0
3000	3600.000	34140	0.0301	**3600.000**	39436	0.0084	3600.000	32221	0.0150
3000	2010.185	19088	0	**655.391**	11914	0	1658.916	18541	0
3000	1448.394	12365	0	**332.774**	4931	0	486.446	7996	0
4000	3600.000	19031	0.0239	**3600.000**	28853	0.0104	3600.000	40321	0.0221
4000	3600.000	21294	0.0246	**3600.000**	23163	0.0208	3600.000	21376	0.0278
4000	**3600.000**	18839	0.0239	**3600.000**	19611	0.0239	3600.000	21079	0.0404
4000	3600.000	19169	0.0288	3600.000	23981	0.0213	**3600.000**	24772	0.0132
4000	3600.000	17197	0.0421	**3600.000**	17233	0.0294	3600.000	17057	0.0439

Table 7.14: Comparison among Algorithm 28, $Var1$ and $Var2$ on instances R_{Grid}

Tables 7.15 and 7.16 show that $Var2$ is the best algorithm on instances with a *Geom* topology. Although the other algorithms are timed out after one hour for instances with 2000 nodes, the variant $Var2$ may solve these instances in minutes. The combination of subtour, cut and multicut inequalities separated by our Algorithm 26 appears to be very effective in this topology with several clusters. Surprisingly, although the MIP Start is very close to the optimal solution, variant $Var1$ is more effective than Algorithm 28.

$\|V\|$	RT_{Algo28}	NN_{Algo28}	Gap_{Algo28}	RT_{Var1}	NN_{Var1}	Gap_{Var1}	RT_{Var2}	NN_{Var2}	Gap_{Var2}
500	1.367	359	0	**0.981**	213	0	1.644	411	0
500	5.341	880	0	2.995	545	0	**1.197**	273	0
500	0.873	158	0	0.992	272	0	**0.590**	149	0
500	1.162	247	0	2.066	396	0	**1.102**	308	0
500	**0.484**	64	0	1.242	203	0	1.432	346	0
1000	31.582	2375	0	37.526	1949	0	**14.134**	1132	0
1000	**87.573**	5732	0	138.934	4973	0	106.967	9366	0
1000	155.944	4795	0	**24.998**	1542	0	66.586	2757	0
1000	253.834	6592	0	218.981	4696	0	**45.923**	2239	0
1000	**20.814**	1618	0	209.993	8087	0	24.832	1644	0
1500	3600.000	76133	0.0097	3600.000	87502	0.0081	**290.099**	6511	0
1500	674.556	18060	0	163.593	2942	0	**43.192**	1993	0
1500	3600.000	116098	0.0064	3600.000	146095	0.0032	**2579.880**	123222	0
1500	835.824	19947	0	370.106	7963	0	**130.060**	17204	0
1500	242.783	4062	0	411.364	9251	0	**69.951**	2050	0
2000	3600.000	80084	0.0073	1379.353	58556	0	**178.782**	3020	0
2000	3600.000	51091	0.0109	3600.000	161770	0.0085	**284.394**	9016	0
2000	3600.000	83185	0.0061	**1264.763**	50473	0	1301.761	57303	0
2000	3600.000	41311	0.0048	3600.000	47187	0.0048	**512.612**	9244	0
2000	3600.000	35847	0.0096	3600.000	401760	0.0024	**430.451**	16907	0
3000	3600.000	34174	0.0058	3600.000	49138	0.0068	**1279.622**	59879	0
3000	3600.000	255362	0.0049	3600.000	62768	$+\infty$	**711.230**	7175	0
3000	3600.000	27174	0.0078	1717.618	154310	0	**306.677**	9380	0
3000	**2622.563**	21984	0	3600.000	50187	0.0029	3600.000	98891	0.0039
3000	3600.000	15984	0.0155	3600.000	90311	0.0097	**3600.000**	94137	0.0019

Table 7.15: Comparison among Algorithm 28, $Var1$ and $Var2$ on instances I_{Geom}

$\|V\|$	RT_{Algo28}	NN_{Algo28}	Gap_{Algo28}	RT_{Var1}	NN_{Var1}	Gap_{Var1}	RT_{Var2}	NN_{Var2}	Gap_{Var2}
500	2.735	745	0	1.179	368	0	**0.767**	233	0
500	0.954	328	0	**0.759**	214	0	1.038	362	0
500	2.879	874	0	1.505	391	0	**0.900**	311	0
500	4.405	1250	0	2.615	678	0	**1.219**	469	0
500	1.550	447	0	1.162	440	0	**0.750**	224	0
1000	97.490	5958	0	264.260	15650	0	**54.465**	3182	0
1000	242.477	15696	0	90.502	3652	0	**19.139**	1600	0
1000	82.940	5091	0	23.335	1582	0	**4.903**	576	0
1000	45.883	2454	0	51.924	3699	0	**31.506**	2220	0
1000	**5.471**	750	0	22.309	2028	0	55.911	.197	0
1500	1578.051	12679	0	262.360	5668	0	**66.672**	2425	0
1500	2566.761	84510	0	1051.018	21326	0	**91.374**	4681	0
1500	3600.000	88810	0.0835	3600.000	90286	0.0508	**5.864**	4320	0
1500	2136.093	74305	0	753.173	13666	0	**87.467**	3038	0
1500	125.283	7293	0	29.285	1476	0	**28.876**	1276	0
2000	3600.000	44623	0.0764	1095.974	22731	0	**1036.856**	22995	0
2000	3600.000	43190	0.1132	3600.000	46120	0.1443	**3600.000**	166288	0.0877
2000	3600.000	38282	0.0721	3600.000	37543	0.0837	**505.104**	9443	0
2000	3600.000	56004	0.0427	3600.000	38349	0.0484	**1937.068**	45123	0
2000	3600.000	45701	0.0972	3600.000	53143	0.0353	**2864.335**	144020	0

Table 7.16: Comparison among Algorithm 28, $Var1$ and $Var2$ on instances R_{Geom}

Tables 7.17 and 7.18 show that Algorithm 28 outperforms its variants in instances with a *Rand* topology. Using a MIP Start, Algorithm 28 solves several instances at the root node. Moreover, in the few cases where the algorithm did not finish in one hour, it ends with a solution with a very small gap. On those dense instances, basic spanning tree's inequalities seem not very effective compared to our inequalities combining both the SPT and the KP.

$\|V\|$	RT_{Algo28}	NN_{Algo28}	Gap_{Algo28}	RT_{Var1}	NN_{Var1}	Gap_{Var1}	RT_{Var2}	NN_{Var2}	Gap_{Var2}
500	**6.789**	1	0	1959.928	29300	0	250.878	4581	0
500	**10.659**	37	0	3213.281	39004	0	713.327	38985	0
500	**13.110**	177	0	123.820	1643	0	1186.426	17760	0
500	**11.602**	1	0	67.352	653	0	3600.000	40847	$+\infty$
500	**56.952**	2220	0	134.689	1761	0	202.142	4926	0
1000	**43.544**	3	0	118.957	327	0	3600.000	4332	$+\infty$
1000	3600.000	675545	0.0027	**759.024**	2063	0	3600.000	4802	$+\infty$
1000	3600.000	411579	0.0027	**153.987**	503	0	3600.000	5335	$+\infty$
1000	**314.165**	5061	0	3600.000	7118	$+\infty$	3600.000	5864	$+\infty$
1000	**43.043**	1	0	3600.000	4547	$+\infty$	3600.000	6627	$+\infty$

Table 7.17: Comparison among Algorithm 28, $Var1$ and $Var2$ on instances I_{Rand}

$\|V\|$	RT_{Algo28}	NN_{Algo28}	Gap_{Algo28}	RT_{Var1}	NN_{Var1}	Gap_{Var1}	RT_{Var2}	NN_{Var2}	Gap_{Var2}
500	1.982	1	0	2.122	1	0	**1.191**	15	0
500	1.349	1	0	2.234	3	0	**0.766**	3	0
500	1.726	1	0	2.971	7	0	**1.513**	19	0
500	1.666	1	0	2.114	7	0	**1.503**	35	0
500	**1.890**	1	0	19.579	95	0	3.496	140	0
1000	**12.170**	1	0	3600.000	6870	$+\infty$	3600.000	6035	$+\infty$
1000	**11.370**	1	0	3600.000	7923	$+\infty$	3600.000	7923	$+\infty$
1000	**15.473**	1	0	136.283	93	0	3600.000	2899	$+\infty$
1000	**13.712**	1	0	378.178	245	0	308.207	716	0
1000	**12.997**	1	0	3600.000	7458	$+\infty$	3600.000	7630	$+\infty$

Table 7.18: Comparison among Algorithm 28, $Var1$ and $Var2$ on instances R_{Rand}

All of these results show that, depending on the topology of the graph, one of the three algorithms performs better. We observe that spanning tree inequalities are efficient for instances with a low density such as *Grid* and *Geom* topologies. Inequalities combining both spanning tree and knapsack problems are efficient for the *Grid* and *Rand* topologies. The results show that using a MIP Start is very efficient only for *Rand* instances. Notice that the three algorithms outperform the subtour and MTZ strategies for the *Grid* and *Geom* topologies and the subtour formulation for the *Rand* instances. Algorithm 28 appears to be the more robust among the three algorithms as it is effective in the three topologies, especially in the *Rand* instances. On the latter instances, Algorithm 28 may solve the problem faster than MTZ formulation, which is particularly effective on this topology. Furthermore, Algorithm 28 guarantees a feasible solution at the end of the procedure.

Chapter 8

Branch-and-Cut for the minimum d-budgeted spanning tree problem and computational experiments

Many real world applications such as telecommunication or road network design require more than a single budget constraint. Let us remember that the resulting generalization of the 1-budgeted to d-budgeted problem is defined as follows:

$$\min \sum_{e \in E} c_e x_e \tag{8.1}$$

$$s.t. \sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subseteq V, 2 \leq |S| \leq |V| - 1, \tag{8.2}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{8.3}$$

$$\sum_{e \in E} w_e^i x_e \leq B^i, \qquad \forall i = 1, \ldots, d, \tag{8.4}$$

$$x_e \in \{0, 1\}, \qquad \forall e \in E. \tag{8.5}$$

The dBSTP is a difficult occurrence of integer programming for several reasons. In contrast with the 1-budgeted case and the multidimensional KP, checking the feasibility is NP-hard for $d \geq 2$. For instance, $x_e = 0,\ \forall e \in E$ is a feasible solution for the KP. Therefore, providing a good initial feasible solution to the Branch-and-Cut algorithm is a challenging task. Furthermore, the case $d \geq 2$ imposes to consider in the disjunctive program proposed for $d = 1$, the NP-hard problem of the intersection of $d + 1$ matroids. In opposition to the case $d = 1$, the extended formulation discussed in Chapter 5 is not anymore exact. The surrogate relaxation is a classical technique allowing one to replace d budget constraints by one constraint obtained as a linear combination of the d constraints.

$$\min \sum_{e \in E} c_e x_e \tag{8.6}$$

$$s.t. \sum_{e \in E[S]} x_e \leq |S| - 1, \qquad \forall S \subseteq V, 2 \leq |S| \leq |V| - 1, \tag{8.7}$$

$$(P_\lambda) \quad \sum_{e \in E} x_e = |V| - 1, \tag{8.8}$$

$$\sum_{e \in E} \sum_{i=1}^{d} \lambda_i w_e^i x_e \leq \sum_{i=1}^{d} \lambda_i B^i, \tag{8.9}$$

$$x_e \in \{0, 1\}, \qquad \forall e \in E. \tag{8.10}$$

Any valid inequality for the surrogate problem (P_λ) is also a valid inequality for the d-budgeted problem. It is not an easy task to fix a priori a value of $\lambda \geq 0$ such that the convex hull of P_λ is close to the one of the d-budgeted problem. Experimental tests show that applying the techniques discussed in Chapter 6 to (P_λ) yields weak valid inequalities.

One of the key ingredients of the success of the algorithm for $d = 1$ is the exploitation of the geometry of the Lagrangian relaxation problem to devise strong valid inequalities. However, solving efficiently the Lagrangian relaxation problem for $d \geq 2$ is a challenging task. We sacrifice the quality of the valid inequalities obtained by an exact algorithm for the sake of the speed of a subgradient method.

We discuss in this chapter an efficient algorithm based on Algorithm 28 to solve the MdBSTP. The cutting plane phase relies essentially on heuristics to generate spanning tree or cover inequalities adapted to tackle the d-budget constraints. The ones presented for $d = 1$ would also generate valid inequalities for this case but their effectiveness is limited. The reader is invited to read Chapters 6 and 7 for a detailed description of the separation heuristics before proceeding forward.

8.1 Preprocessing

During the preprocessing step, we generate a number of subtour inequalities by exploiting the Lagrangian relaxation function. By dualizing the d-budget constraints (8.4), the Lagrangian relaxation problem is defined as follows:

$$\max_{z_1,\ldots,z_d \geq 0} L(z) = \min \sum_{e \in E} c_e x_e - \sum_{i=1}^{d} z_i (B^i - \sum_{e \in E} w_e^i x_e) \tag{8.11}$$

$$\sum_{e \in E[S]} x_e \leq |W| - 1, \qquad \forall S \subset V, 2 \leq |S| \leq |V| - 1, \tag{8.12}$$

$$\sum_{e \in E} x_e = |V| - 1, \tag{8.13}$$

$$0 \leq x_e \leq 1, \qquad \forall e \in E. \tag{8.14}$$

The Lagrangian function L is piecewise linear and concave. Since the spanning tree polytope, defined by constraints (8.12)-(8.14) is exact, it follows that the optimal value

of the Lagrangian relaxation coincides with the optimal value of the linear relaxation of problem (8.1)-(8.5). In contrast to the case $d = 1$, we solve approximately problem (8.11)-(8.14) by a subgradient technique.

One of the difficulties of the case $d \geq 2$ is to generate a good feasible solution to feed the Branch-and-Cut algorithm. The preprocessing algorithm starts building such a solution by calling the heuristic Algorithm 29. The idea is to generalize the binary search in the case $d = 1$ to higher dimensions. The algorithm explores iteratively the parametric space $z_1, \ldots, z_d$. At each iteration, it selects a vector $(z_1, \ldots, z_d)$ and computes a minimum spanning tree with composite costs $c_e + \sum_{i=1}^{d} z_i w_e^i$. The best feasible solution, if any, is outputted at the end. Computational results show that the heuristic is fast and generally succeeds in finding a feasible solution.

Algorithm 29: Heuristics - Computation of a tight feasible solution

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}_+^{d \times |E|}$, $B \in \mathbb{N}^d$, $lb, ub \in \mathbb{R}_+^d, 0 \leq lb \leq ub, \epsilon \in \mathbb{R}$

```
T* ← ∅;
z* ← ub;
z ← 0^d;
while Σ_{i=1}^{d} ub_i − lb_i > ε do
    for i = 1, ..., d do
        z_i ← (lb_i + ub_i)/2;
    end
    Compute the minimum spanning tree T = (V, F) of cost Σ_{e∈F}(c_e + Σ_{i=1}^{d} z_i w_e^i);
    for i = 1, ..., d do
        if Σ_{e∈F} w_e^i > B^i then
            lb_i ← z_i;
        else
            ub_i ← z_i;
        end
    end
    if T is a feasible solution then
        if T* == ∅ or Σ_{e∈E[T*]} c_e > Σ_{e∈F} c_e then
            T* ← T;
            z* ← z;
        end
    end
end
returns T*, z*;
```

In the case $d = 1$, the Lagrangian relaxation produces a good feasible solution corresponding to a minimum spanning tree for the composite costs $c_e + z^* w_e$, where z^* is a maximizer of the Lagrangian function. The feasible solution computed in Algorithm 29 may correspond to a minimum spanning tree for a composite cost $c_e + \sum_{i=1}^{d} \overline{z}_i w_e^i$ where vector $(\overline{z}_1, \ldots, \overline{z}_d)$ may be far from the maximizer $(z_1^*, \ldots, z_d^*)$ of the Lagrangian function. In order to improve the quality of the obtained solution, Algorithm 30 implements a subgradient procedure in order to move from $(\overline{z}_1, \ldots, \overline{z}_d)$ to a vector closer to $(z_1^*, \ldots, z_d^*)$.

The algorithm iterates $z^{k+1} = z^k + \alpha \nabla_{z^k}$ [13]. Here z^k is the k-th iterate and ∇_{z^k} is the gradient of function L at z^k.

Algorithm 30: Subgradient algorithm

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{d \times |E|}$, $B \in \mathbb{N}^d$, $T^* = (V, F^*)$, $z^* \in \mathbb{R}^d$, $K, \alpha \in \mathbb{R}^+$

$z \leftarrow z^*$:
for $j \in \{1, \ldots, K\}$ **do**
  Compute the minimum spanning tree $T = (V, F)$ of cost $\sum_{e \in F}(c_e + \sum_{i=1}^{d} z_i w_e^i)$;
  for $i = 1, \ldots, d$ **do**
    $z_i \leftarrow z_i + \alpha(\sum_{e \in F} w_e^i - B^i)$;
  end
  if T is a feasible solution **then**
    if $T^* == \emptyset$ or $\sum_{e \in F^*} c_e > \sum_{e \in F} c_e$ **then**
      $T^* \leftarrow T$;
      $z^* \leftarrow z$;
    end
  end
end
Return T^*, z^*;

Similarly to the case $d = 1$, once a good feasible solution is computed, the algorithm generates a number of subtour inequalities at the root node of the Branch-and-Cut algorithm by performing edge swaps. All the preprocessing procedures are summarized in Algorithm 31. Computational results show that the generated subtour inequalities are useful to improve the running time.

Algorithm 31: Generation of subtour inequalities

Data: $G = (V, E)$, $c \in \mathbb{R}_+^{|E|}$, $w \in \mathbb{N}^{d \times |E|}$, $B \in \mathbb{N}^d$, $lb, ub \in \mathbb{R}_+^d, 0 \leq lb \leq ub$

\# **Find a tight feasible solution**
$T^*, z^* \leftarrow$Algorithm 29$(G, c, w, B, lb, ub, 10^{-3})$;
\# **Improve** T^* **with a subgradient algorithm**
$T^* = (V, F^*), z^* \leftarrow$ Algorithm 30$(G, c, w, B, T^*, z^*, 100, 10^{-7})$;
if $T^* \neq \emptyset$ **then**
  $M \leftarrow \max\{c_e + \sum_{i=1}^{d} z_i^* w_e^i \mid e \in F^*\}$;
  for $e \in E \backslash F^*$ **do**
    if $c_e + \sum_{i=1}^{d} z_i^* w_e^i \leq M$ **then**
      Add the subtour constraint $\sum_{e \in E[W]} x_e \leq |W| - 1$, where $W \subset V$ contains all vertices of the created cycle by adding e to T^*;
    end
  end
end

8.2 Cutting plane algorithms and heuristics

Like the case $d = 1$, we implement a cutting plane method in order to iteratively define the set of feasible solutions by means of spanning tree and knapsack valid inequalities.

8.2.1 Separation of spanning tree's valid inequalities

By exploiting the structure of the encountered solutions during the search, we propose efficient heuristics to separate valid inequalities for the spanning tree polytope in the same vein as the case $d = 1$. Computational experiments show that the heuristics for the case $d = 1$ need to be adapted to handle the case $d \geq 2$. Similarly to the 1BSTP, in the first nodes of the search tree, the encountered solutions x^* are integer but their support graph $G^* = (V, E^*)$, where $E^* = \{e \in E \mid x^*_e > 0\}$, are not connected. Since one of the connected components contains at least one cycle, Algorithm 32 adds subtour constraints. The formulation is strengthened by also adding cut inequalities induced by the connected components. Algorithm 32 requires $O(|V||E|)$ time but is very effective in practice.

Algorithm 32: Separation algorithm for a given integer solution

Data: $E, x^* \in \{0,1\}^{|E|}, G^* = (V, E^*)$
if there are $p > 1$ connected components in G^* **then**
  for every connected component $G^*[V_i]$, $i = 1, \ldots, p$ **do**
    if component $G^*[V_i]$ contains more than $|V_i| - 1$ edges **then**
      Add the subtour constraint $\sum_{e \in E[V_i]} x_e \leq |V_i| - 1$;
    end
  end
  if $p > 2$ **then**
    for every connected component $G^*[V_i]$, $i = 1, \ldots, p$ **do**
      Add the cut constraint $\sum_{u \in V_i, v \in V \setminus V_i, \{u,v\} \in E} x_{\{u,v\}} \geq 1$;
    end
  end
end

Algorithm 33 separates fractional solutions x^* such that their support graph G^* is not connected. As shown for the case $d = 1$, at least one subtour inequality is violated in one connected component $G^*[V_i]$. Consequently, the algorithm checks all the components and adds the violated inequalities. Furthermore, it explores each connected component $G^*[V_i]$ and checks if a subtour inequality is violated over $G^*[V_i]$ after deleting all the branches. Finally, the algorithm considers the partial graph obtained from G^* after deleting all the branches with fractional edge, if any. In the positive case, the algorithm adds a multicut inequality to cut off x^*. In the worst case, Algorithm 33 runs in $O(|V||E|)$ but is very effective in practice.

Algorithm 33: Separation algorithm for a fractional solution when G^* is not connected

Data: E, $x^* \in [0:1]^{|E|}, G^* = (V, E^*)$ where $E^* = \{e \in E | x^*(e) > 0\}$

for every connected component $G^*[V_i]$, $i = 1, \ldots, p$ **do**
  if $\sum_{\{u,v\} \in E^* | u,v \in V_i} x_{\{u,v\}} > |V_i| - 1$ **then**
    Add the subtour constraint $\sum_{\{u,v\} \in E | u,v \in V_i} x_{\{u,v\}} \leq |V_i| - 1$;
  end
  Let F_i be the set of edges in branches in $G^*[V_i]$;
  if $\sum_{\{u,v\} \in E^* \setminus F_i | u,v \in V_i} x_{\{u,v\}} > |V_i| - 1 - |F_i|$ **then**
    Add the subtour constraint $\sum_{\{u,v\} \in E \setminus F_i | u,v \in V_i} x_{\{u,v\}} \leq |V_i| - 1 - |F_i|$;
  end
end
Consider the subgraph $G' = (V, E')$ where $E' = E^* \setminus \{e \in E^* \mid x^*_e < 1$ and e belongs to a branch of $G^*\}$;
Consider the partition $V_1, \ldots, V_{p'}$ of V, built with the connected components of G';
if $p' > 2$ **then**
  Add the multicut constraint $\sum_{\{u,v\} \in E | u \in V_i, v \in V_j, i \neq j} x_{\{u,v\}} \geq p' - 1$;
end

Algorithm 34 handles, like the case $d = 1$, the remaining situation where the encountered solution x^* is fractional and its support graph G^* is connected. In the worst case, Algorithm 34 runs in $O(|V|^2 + |E|)$.

Algorithm 34: Separation algorithm for a fractional solution when G^* is connected

Data: E, $x^* \in [0:1]^{|E|}, G^* = (V, E^*)$ where $E^* = \{e \in E | x^*(e) > 0\}$

Consider the subgraph $G' = (V, E')$ where $E' = E^* \setminus \{e \in E^* \mid x^*_e < 1$ and e belongs to a branch of $G^*\}$;
if $|\{e \in E^* \mid x^*_e < 1$ and e belongs to a branch of $G^*\}| > 0$ **then**
  Consider the partition $V_1, \ldots, V_{p'}$ of V, built with the connected components of G';
  Add the multicut constraint $\sum_{\{u,v\} \in E | u \in V_i, v \in V_j, i \neq j} x_{\{u,v\}} \geq p' - 1$;
end

All the resulting inequalities are lifted, similarly to the case $d = 1$, by considering not only the edges in the support graph G^* but also the edges in the original graph G involved in the subtour or the cut inequalities.

8.2.2 Separation of cover inequalities

We generalize in this section the heuristics for separating cover inequalities in the case $d = 1$ to tackle more budget constraints. The idea is to solve heuristically an interdiction problem for each budget constraint $\sum_{e \in E} w^i_e x_e \leq B^i,\ i = 1, \ldots, d$. Consider a fractional solution x^* and its support graph $G^* = (V, E^*)$. Recall that the goal is to compute a subset

$R^i \in E^*$ satisfying the following conditions:

1. no spanning tree in $G^i = (V, E^* \backslash R^i)$ satisfies the budget constraint $\sum_{e \in E} w_e^i x_e \leq B^i$, and,
2. $\sum_{e \in R^i} x_e^* < 1$.

Let $T^* = (V, F^*)$ denotes a spanning tree maximizing $\sum_{e \in F^*} x_e^*$ in G^*. Algorithm 35 sorts the edges in $E^* \backslash F^*$ in non decreasing w_e^i values and returns a maximal subset R^i satisfying condition (2).

Algorithm 35: Heuristic - Interdiction spanning tree problem

```
Data: G = (V, E), E*, w^i ∈ ℕ^|E|, x* ∈ [0;1]^|E|, T = (V, F)
L ← Sort E*\F in a non-decreasing order of w^i;
j ← 1;
Z ← ∅;
xz ← 0;
while j ≤ |E*\F| do
    if xz + x*(L[j]) < 1 then
        Z ← Z ∪ {L[j]};
        xz ← xz + x*(L[j]);
    end
    j ← j + 1;
end
return G^i = (V, E*\Z);
```

If G^i satisfies conditions (1) and (2), then $\sum_{e \in E^* \backslash R^i} x_e \leq |V| - 2$ is a valid inequality that cuts off x^*. The algorithm lifts it using Algorithm 14 and adds it to the model. Otherwise, although there is a spanning tree in G^i that satisfies the constraint $\sum_{e \in E} w_e^i x_e \leq B^i$, due to the conflicts between the budget constraints, G^i may not contain any solution that satisfies simultaneously all the budget constraints. The following integer program, based on the MTZ formulation, solves the related feasibility problem. We introduce a new variable $y \in \{0, 1\}$ such that $y = 0$ if there is at least one feasible solution in G^i, $y = 1$ otherwise. Although it is NP-hard to solve the *feasibility problem*, by choosing a particular objective function, where M is a constant of high value ($M > \max\{B^i \mid i = 1, \ldots, d\}$), the following ILP can be solved very fast in practice and can then be used as a subroutine. Solving the ILP (8.15)-(8.24) ensures that the inequality $\sum_{e \in E \backslash R^i} x_e \leq |V| - 2$ is valid in this case.

$$\min \quad My \tag{8.15}$$

$$s.t. \sum_{i \in \delta^-(j)} x_{ij} = 1, \qquad \forall j \in V \backslash \{r\}, \tag{8.16}$$

$$u_i - u_j + |V| x_{ij} \leq |V| - 1, \qquad \forall (i,j) \in A, j \neq r, \tag{8.17}$$

$$x_{ij} + x_{ji} \leq 1, \qquad \forall \{i,j\} \in E, \tag{8.18}$$

$$\sum_{ij \in A} w_{ij}^l x_{ij} - My \leq B^l, \qquad l = 1, \ldots, d, \tag{8.19}$$

$$u_i \leq |V| - 1, \qquad \forall i \in V \backslash \{r\}, \tag{8.20}$$

$$u_i \geq 1, \qquad \forall i \in V \backslash \{r\}, \tag{8.21}$$

$$u_r = 0, \tag{8.22}$$

$$y \in \{0,1\}, \tag{8.23}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in V, j \in V \backslash \{i,r\}. \tag{8.24}$$

The separation procedure is summarized in algorithm 36.

Algorithm 36: Heuristic - Separation algorithm for cover inequalities

Data: $G = (V,E), c \in \mathbb{R}_+^{|E|}, w \in \mathbb{N}^{d \times |E|}, B \in \mathbb{N}^d, x^* \in [0;1]^{|E|}$

Let $G^* = (V, E^*)$ be such that $E^* = \{e \in E \mid x_e^* > 0\}$;
Let $T^* = (V, F)$ be the spanning tree maximizing $\sum_{e \in F} x_e^*$ in G^*;
for $i = 1, \ldots, d$ **do**
  if $\sum_{e \in F} w_e^i > B^i$ **then**
    # **Searching a subgraph of G^* with no feasible solution**
    $G^i = (V, S) \leftarrow$ Algorithm 35 (G, E^*, w^i, x^*, T^*);
    Let $T^i = (V, F^i)$ be the spanning tree minimizing $\sum_{e \in F^i} w_e^i$ in G^i;
    if $\sum_{e \in F^i} w_e^i > B^i$ **then**
      Lift the inequality $x(S) \leq |V| - 2$ using Algorithm 14 $(G, w^i, B^i, G^i = (V, S))$ and add it to the model;
    else
      # **Solve the feasibility problem**
      Solve the ILP (8.15)-(8.24) related to G^i and let (x^*, u^*, y^*) be the optimal solution;
      if $y^* = 1$ **then**
        Add the inequality $x(S) \leq |V| - 2$ to the model;
      end
    end
  end
end

8.3 Algorithm for the minimum d-budgeted spanning tree problem

In this Section, we present an algorithm for the MdBSTP combining all of the previous procedures.

Algorithm 37: Optimization procedure for MdBSTP

Data: $G = (V, E), w \in \mathbb{N}^{d\times|E|}, c \in \mathbb{R}_+^{|E|}, B \in \mathbb{N}^d$

Build the Integer Linear Problem $\min\{\sum_{e\in E} c_e x_e \mid \sum_{e\in E} x_e = |V| - 1, \sum_{e\in E} w_e^i x_e \leq B^i, i = 1, \ldots, d, x \in \{0,1\}^{|E|}\}$;

 # Add subtour constraints to the root

Run Algorithm 31 (G, c, w, B, lb, ub);

 # Calling the Branch-and-Cut algorithm

Solve the ILP by Branch-and-Cut, with the separation algorithm for subtour and cover inequalities 38 (G, x^*, w, c, B);

The final heuristic separation algorithm is given below. We set $K = 40|V|$.

Algorithm 38: Separation algorithm for MdBSTP

Data: $G = (V, E)$, $x^* \in [0:1]^{|E|}, w \in \mathbb{N}^{d\times|E|}, c \in \mathbb{R}_+^{|E|}, B \in \mathbb{N}^d$

Consider the subgraph $G^* = (V, E^*)$ where $E^* = \{e \in E \mid x^*(e) > 0\}$;
if x^* is integer **then**
  Run Algorithm 32 (E, x^*, G^*)
else
  if the total number of separation procedure is lower than K **then**
    if there are $p > 1$ connected components in G^* **then**
      Run Algorithm 33$(E,\ x^*, G^*)$;
    else
      Run Algorithm 34 $(E,\ x^*, G^*)$;
      Run Algorithm 36 (G', c, w, B, x^*);
    end
  end
end

8.4 Computational experiments

The difficulty of the MdBSTP relies on the number d of budget constraints. In that sense, we present a computational study with the values $d = 2, 5$ and 10. For each edge $e \in E$, the weights $w_e^i,\ i = 1, \ldots, d$ follow a uniform law in $[0:100]$. We consider a correlated conflicting cost $c_e = 100 - \frac{\sum_{i=1}^d w^i}{d} + a$ where a is a uniform random value in [0;20]. For every instance, we fix the budgets $B^i = 40(|V| - 1),\ i = 1, \ldots, d$. We consider the same graph topologies as the $d = 1$ case (see section 7.1). In order to illustrate the efficiency of

Algorithm 37, we will compare its resolution time with the subtour and MTZ formulations adapted from section 7.2. The results are presented in the following Tables, we used the same metrics as section 7.6.

Table 8.1 reports the performances of the three algorithms in instances *Grid* with $d = 2$. Computational results show that our algorithm outperforms the subtour and MTZ formulation in these instances. Similarly to the $d = 1$ case, the MTZ formulation is dominated by the two other strategies and it times out after one hour for moderate instances of 1000 nodes and higher (without even finding a feasible solution from 3000 vertices). The subtour formulation is systematically timed out from 3000 nodes (and fails to find a feasible solution in one hour). Algorithm 37 solves at optimality all but two instances that timed out with a very small gap (0.0005%). Although the cost and weight values generation is not the same as the $d = 1$ case, we manage to solve at optimality higher instances for $d = 2$ than the $d = 1$ case.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	0.889	855	0	**0.266**	302	0	23.583	110336	0
100	1.022	1079	0	**0.683**	731	0	1.878	2962	0
100	2.268	2098	0	**0.644**	1185	0	1.393	3537	0
100	1.488	1917	0	**0.672**	1189	0	4.754	34703	0
100	**0.435**	307	0	0.957	1351	0	4.828	28897	0
500	20.776	2878	0	**20.523**	6199	0	34.549	33713	0
500	11.957	713	0	**5.421**	1241	0	225.222	349287	0
500	6.655	438	0	**1.136**	264	0	270.629	342243	0
500	9.219	1590	0	**2.368**	1042	0	468.848	677974	0
500	13.411	2213	0	**7.667**	1867	0	3600.000	2965193	0.0085
1000	151.757	1848	0	**12.741**	1377	0	3600.000	2007514	0.0107
1000	102.947	1127	0	**25.892**	2529	0	3600.000	1887212	0
1000	77.604	1116	0	**35.888**	2737	0	3600.000	1371863	$+\infty$
1000	273.447	1544	0	**21.140**	1407	0	1140.635	597926	0
1000	105.849	1345	0	**18.794**	2134	0	3600.000	1436283	$+\infty$
1500	593.396	1502	0	**40.545**	2587	0	3600.000	1307596	0.0456
1500	568.123	2160	0	**20.160**	1392	0	3600.000	1982254	0.0870
1500	407.213	1302	0	**28.019**	2188	0	3600.000	1746118	0.0637
1500	587.617	1658	0	**83.645**	3867	0	3600.000	1457059	$+\infty$
1500	412.148	4881	0	**64.281**	3658	0	3600.000	1634017	0.0454
2000	2659.903	1058	0	**273.213**	8113	0	3600.000	1064432	0.1045
2000	1919.369	5476	0	**65.705**	2882	0	3600.000	856524	0.1156
2000	1665.449	1030	0	**194.746**	5678	0	3600.000	1147023	0.1207
2000	3067.207	1071	0	**305.287**	8632	0	3600.000	975460	$+\infty$
2000	1759.328	2292	0	**141.031**	3010	0	3600.000	1003478	0.1508
3000	3600.000	1158	$+\infty$	**158.477**	3081	0	3600.000	137630	$+\infty$
3000	3600.000	715	$+\infty$	**905.855**	9843	0	3600.000	164364	$+\infty$
3000	3600.000	905	$+\infty$	**812.016**	7374	0	3600.000	210708	$+\infty$
3000	3600.000	1106	$+\infty$	**364.876**	5036	0	3600.000	164501	$+\infty$
3000	3600.000	1045	$+\infty$	**945.676**	9166	0	3600.000	278018	$+\infty$
4000	3600.000	5336	$+\infty$	**3600.000**	21832	0.0005	3600.000	104946	$+\infty$
4000	3600.000	5207	$+\infty$	**3600.000**	10823	0.0005	3600.000	104513	$+\infty$
4000	3600.000	869	$+\infty$	**1215.001**	6930	0	3600.000	110846	$+\infty$
4000	3600.000	745	$+\infty$	**346.391**	3219	0	3600.000	103008	$+\infty$
4000	3600.000	803	$+\infty$	**2313.501**	12822	0	3600.000	110853	$+\infty$

Table 8.1: Instances *Grid* with $d = 2$

Table 8.2 shows that Algorithm 37 also outperforms the other algorithms on instances *Grid* with $d = 5$. Note that combining the *Grid* topology and the 5 budget constraints often leads to infeasible instances, therefore for these instances, we set the budgets to $B^i = 50(|V| - 1),\ \forall i = 1, \ldots, d$ (instead of $40(|V| - 1)$. Table 8.2 shows that the subtour and MTZ formulations are systematically timed out in instances with even 300 nodes. Algorithm 37 solves instances up to 600 vertices.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	**12.049**	6852	0	12.884	6328	0	786.616	4231268	0
100	59.112	78204	0	**17.534**	172928	0	3600.000	20248990	0.0017
100	153.994	222762	0	**15.357**	147668	0	341.355	1057279	0
100	31.491	44736	0	**9.017**	54095	0	1253.631	3023104	0
100	39.876	67743	0	**12.571**	90900	0	71.500	277286	0
200	1261.865	1012659	0	**109.580**	1079894	0	3600.000	3267166	0.1282
200	3600.000	2598154	0.0282	**229.230**	2205142	0	1800.112	4317842	0
200	2997.634	1597072	0	**704.211**	2941995	0	3600.000	4124249	0.0421
200	1074.119	723211	0	**106.008**	722766	0	3600.000	11127094	0.1551
200	657.371	338948	0	**53.813**	274745	0	1427.989	2436301	0
300	3600.000	1303033	0.0187	**1521.693**	3527060	0	3600.000	4260219	0.0094
300	3222.526	1410422	0	**260.352**	2318920	0	3600.000	4898259	0.0057
300	3600.000	965446	0.0188	**275.055**	1735909	0	3600.000	7656042	0.1220
300	3600.000	1423852	0.0985	**189.412**	768207	0	3600.000	4403862	0.0038
300	3600.000	1274862	0.4530	**205.920**	1331733	0	3600.000	5007462	0.0149
400	3600.000	1422916	0.0142	**915.761**	5317597	0	3600.000	3312405	0.1048
400	3600.000	473016	0.0282	**323.467**	1536833	0	3600.000	6073722	0.1064
400	3600.000	1342856	0.0750	**434.521**	2394931	0	3600.000	4050489	0.1023
400	3600.000	1228904	0.0258	**1042.158**	74914010	0	3600.000	2235016	0.1770
400	3600.000	807553	0.1407	**371.759**	1687687	0	3600.000	3770183	0.0328
500	3600.000	749130	0.0114	**671.609**	4309429	0	3600.000	2930713	0.0685
500	3600.000	1258173	0.0113	**950.808**	4766911	0	3600.000	2961623	0.0397
500	3600.000	875053	0.0230	**279.146**	634515	0	3600.000	3142007	0.0845
500	3600.000	993007	0.0751	**518.824**	2155709	0	3600.000	2881056	0.1051
500	3600.000	946077	0.0113	**754.515**	1959318	0	3600.000	3303484	0.0155
600	3600.000	728132	0.0142	**3600.000**	11993124	0.0047	3600.000	2273927	0.0285
600	3600.000	309474	0.0237	**1228.054**	7617825	0	3600.000	2543452	0.0522
600	3600.000	525718	0.0227	**913.463**	6145719	0	3600.000	2201379	0.0684
600	3600.000	398007	0.1217	**3600.000**	20708921	0.0095	3600.000	2890521	0.0423
600	3600.000	667089	0.0182	**3600.000**	6060610	0.0095	3600.000	2220563	0.1671

Table 8.2: Instances *Grid* with $d = 5$

Table 8.3 reports the performance of the three algorithms in instances *Grid* with $d = 10$. Similarly to the $d = 5$ case, we set $B^i = 50(|V| - 1),\ i = 1, \ldots, d$ for these instances. Algorithm 37 succeeds in solving more instances with 100 vertices than the two other algorithms. Although Algorithm 37 times out for instances with 100 or 200 nodes, it ends

with the smallest gap after one hour of computation.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	3600.000	3631954	0.2501	**3600.000**	33803089	0.1945	3600.000	5350463	0.5276
100	1650.030	1670188	0	**382.222**	2619584	0	3600.000	6428532	0.0551
100	3600.000	3972023	0.1117	**1875.396**	18458826	0	3439.252	7936247	0
100	3600.000	2694018	0.1381	**1804.286**	17522451	0	3600.000	6197738	0.2486
100	3600.000	2694018	0.2722	**3600.000**	40161704	0.1634	3600.000	7075803	0.2178
200	3600.000	1040893	0.2537	**3600.000**	7152466	0.0970	3600.000	6658415	0.2960
200	3600.000	636136	0.2117	**3600.000**	10412660	0.0985	3600.000	4231075	0.0989
200	3600.000	1030925	0.1955	**3600.000**	10113714	0.1560	3600.000	41241243	0.2931
200	3600.000	754284	0.2108	**3600.000**	10330309	0.1370	3600.000	3766655	0.3789
200	3600.000	847503	0.2018	**3600.000**	10131788	0.1965	3600.000	3891742	0.3452

Table 8.3: Instances *Grid* with $d = 10$

Tables 8.1, 8.2 and 8.3 show that increasing the number of budget constraints makes the budgeted problems more difficult to solve in instances *Grid*. Indeed, with $d = 2$ Algorithm 37 manages to solve instances up to 4000 nodes, it solves instances up to 600 vertices when $d = 5$ and instances with 100 nodes with $d = 10$.

Table 8.4 reports the computational results of the three algorithms in random geometric instances *Geom* with $d = 2$.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	3.271	2379	0	**0.951**	949	0	25.554	49583	0
100	3.865	5536	0	**1.432**	2985	0	3600.000	5631226	0.0695
100	**0.941**	1322	0	1.399	3255	0	1838.128	3172724	0
100	**0.302**	206	0	0.385	189	0	37.721	61854	0
100	0.220	115	0	**0.124**	15	0	13.721	61854	0
500	108.559	6951	0	**5.304**	704	0	3600.000	2568811	0.0983
500	49.760	2882	0	**10.180**	1577	0	3600.000	2970861	0.0465
500	160.881	15390	0	**12.369**	2145	0	3600.000	2516706	0.0657
500	62.442	2377	0	**8.974**	1356	0	2334.679	1831969	0
500	24.745	1326	0	**10.982**	1570	0	3600.000	2679375	0.0453
1000	1084.856	1032	0	**77.343**	3937	0	3600.000	913814	0.1393
1000	605.096	1034	0	**55.956**	2348	0	3600.000	1205889	0.1263
1000	404.821	840	0	**36.218**	1620	0	3600.000	963875	0.1425
1000	383.090	1045	0	**115.552**	3986	0	3600.000	1072659	0.1254
1000	830.049	9282	0	**55.360**	2107	0	3600.000	1132409	0.0631
1500	2781.302	1041	0	**270.703**	8437	0	3600.000	340889	$+\infty$
1500	3600.000	1453	0.2130	**101.281**	1620	0	3600.000	333133	$+\infty$
1500	2876.916	1065	0	**196.167**	3500	0	3600.000	297810	$+\infty$
1500	3600.000	1060	$+\infty$	**1090.477**	15169	0	3600.000	370540	$+\infty$
1500	2964.898	1386	0	**142.272**	2807	0	3600.000	337859	$+\infty$
2000	3600.000	586	$+\infty$	**2637.661**	31124	0	3600.000	254619	$+\infty$
2000	3600.000	994	$+\infty$	**455.537**	4530	0	3600.000	189005	$+\infty$
2000	3600.000	1037	$+\infty$	**622.820**	8019	0	3600.000	297820	$+\infty$
2000	3600.000	3873	$+\infty$	**1379.325**	9863	0	3600.000	218732	$+\infty$
2000	3600.000	4006	$+\infty$	**1280.786**	14947	0	3600.000	224907	$+\infty$
2500	3600.000	3797	$+\infty$	**642.091**	4172	0	3600.000	144553	$+\infty$
2500	3600.000	1095	$+\infty$	**1327.843**	9703	0.0176	3600.000	97070	$+\infty$
2500	3600.000	4273	$+\infty$	**1347.073**	8292	0	3600.000	110276	$+\infty$
2500	3600.000	3712	$+\infty$	**1499.109**	11502	0	3600.000	137801	$+\infty$
2500	3600.000	1218	$+\infty$	**3600.000**	13027	0.0027	3600.000	98643	$+\infty$
3000	3600.000	6539	$+\infty$	**3600.000**	9032	0.0116	3600.000	50386	$+\infty$
3000	3600.000	1180	$+\infty$	**1537.256**	4715	0	3600.000	72876	$+\infty$
3000	3600.000	1045	$+\infty$	**3600.000**	5917	0.0996	3600.000	64238	$+\infty$
3000	3600.000	2461	$+\infty$	**3600.000**	9319	0.0047	3600.000	54027	$+\infty$
3000	3600.000	6270	$+\infty$	**3600.000**	13045	0.0008	3600.000	60487	$+\infty$

Table 8.4: Instances *Geom* with $d = 2$

Similarly to the $d = 1$ case, the MTZ formulation is systematically timed out from instances with 500 nodes (and it doesn't find any feasible solution from 1500 vertices). The subtour formulation is also systematically timed out from 2000 nodes with no feasible solution found. Algorithm 37 outperforms the two other strategies, and solves at optimality instances up to 3000 nodes.

Table 8.5 illustrates the efficiency of Algorithm 37 in instances *Geom* with $d = 5$. The

subtour and MTZ formulations are timed out with even 200 nodes, whereas Algorithm 37 solves instances up to 300 vertices.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	1093.873	1129025	0	**120.889**	1390678	0	3600.000	5906048	0.1587
100	74.490	51216	0	**10.124**	39769	0	194.767	509169	0
100	152.897	294856	0	**24.070**	274929	0	1166.091	3214108	0
100	1867.197	2152873	0	**150.861**	1680861	0	3600.000	6008794	0.2875
100	853.892	736934	0	**64.554**	720721	0	3600.000	15650971	0.6684
200	3600.000	1709405	0.0454	**1478.325**	10509288	0	3600.000	6141022	0.2158
200	3600.000	1173927	0.0670	**934.776**	2981808	0	1976.903	3514911	0
200	3600.000	1919725	0.0341	**1263.279**	10665128	0	3600.000	4040132	0.1591
200	3600.000	1997980	0.0437	**168.654**	1144784	0	3600.000	8856269	0.1028
200	3600.000	1631447	0.0115	**962.769**	3645021	0	3600.000	6070001	0.1491
300	3600.000	474461	0.0382	**3600.000**	18648682	0.0152	3600.000	3624735	0.0229
300	3600.000	789822	0.0380	**3600.000**	14319962	0.0076	3600.000	2865031	0.0988
300	3600.000	1258732	0.0379	**3600.000**	7043941	0.0153	3600.000	2976689	0.0379
300	3600.000	816727	0.0230	**344.562**	1462993	0	3600.000	3015517	0.1685
300	3600.000	704558	0.0223	**3600.000**	8584058	0.0076	3600.000	2823992	0.2156

Table 8.5: Instances *Geom* with $d = 5$

Table 8.6 shows that Algorithm 37 dominates the other strategies in *Geom* instances with $d = 10$. No instance with 100 nodes has been solved at optimality by the three algorithms, but Algorithm 37 ends with the smallest gap.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
50	1189.158	1148	0	**278.819**	3762240	0	2391.143	7398552	0
50	3600.000	3853373	1.0022	**1456.352**	16676370	0	3600.000	5485777	0.8279
50	306.503	452049	0	**199.183**	2117266	0	766.603	7401415	0
50	2379.681	4246194	0	**219.572**	3490999	0	2887.005	19452343	0
50	**230.609**	604412	0	291.114	3137978	0	3600.000	24065727	$+\infty$
100	3600.000	2184243	0.9880	**3600.000**	9127073	0.4895	3600.000	14008877	$+\infty$
100	3600.000	1278937	0.9640	**3600.000**	23809649	0.4086	3600.000	5024198	0.4946
100	3600.000	1751694	$+\infty$	**3600.000**	18281446	0.9941	3600.000	11937012	$+\infty$
100	3600.000	3454212	0.7002	**3600.000**	20165609	0.4042	3600.000	12401046	1.0807
100	3600.000	2184214	0.9644	**3600.000**	14538856	0.4843	3600.000	6121921	0.5264

Table 8.6: Instances *Geom* with $d = 10$

Similarly with the *Grid* topology, Tables 8.4, 8.5 and 8.6 show that increasing the number of budget constraints makes instances more difficult to solve. In the *Geom* topology, we manage to solve instances up to 3000 vertices with $d = 2$, 300 nodes with $d = 5$, while we solve instances with 50 vertices with $d = 10$.

Table 8.7 shows the results in random dense instances *Rand* with $d = 2$.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
100	11.989	3308	0	**1.226**	76	0	1.543	1113	0
100	6.904	1251	0	0.885	35	0	**0.649**	1	0
100	2.967	1117	0	0.871	78	0	**0.620**	1	0
100	**0.606**	94	0	0.898	41	0	1.136	1	0
100	6.133	1509	0	7.496	2273	0	**1.199**	1	0
300	584.320	25484	0	365.841	3905	0	**12.158**	37	0
300	3600.000	69879	0.1099	620.490	29565	0	**5.871**	1	0
300	522.665	3701	0	373.985	7802	0	**10.869**	1	0
300	373.316	4841	0	285.586	2631	0	**16.689**	3023	0
300	311.428	2225	0	152.893	961	0	**10.589**	1349	0
500	3600.000	4304	$+\infty$	3600.000	2282	$+\infty$	**19.041**	1	0
500	3600.000	4329	$+\infty$	3600.000	2325	$+\infty$	**22.153**	1	0
500	3600.000	4907	$+\infty$	3600.000	2299	$+\infty$	**18.003**	1	0
500	3600.000	4323	$+\infty$	3600.000	2186	$+\infty$	**22.089**	1	0
500	3600.000	4479	$+\infty$	3600.000	2325	$+\infty$	**16.376**	1	0

Table 8.7: Instances *Rand* with $d = 2$

Similarly to the $d = 1$ case, random graphs appear to be particularly challenging, as both the subtour strategy and Algorithm 37 are timed out after one hour and did not manage to find a feasible solution on graphs with even 500 nodes. However, the MTZ formulation solves these instances at the root node in seconds. Recall that the MTZ formulation is also very effective when $d = 1$.

Table 8.8 shows that Algorithm 37 dominates the MTZ formulation when $d = 5$. Algorithm 37 solves instances *Rand* up to 100 nodes whereas the two other algorithms are timed out after one hour.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
50	1992.168	3785517	0	**48.935**	1035506	0	3600.000	28285041	0.1997
50	3600.000	4581780	0.0502	**724.345**	15096146	0	3600.000	18807896	0.2505
50	1055.502	1314188	0	**63.428**	1548164	0	193.310	1048555	0
50	3600.000	6338949	0.1003	**1777.005**	48298607	0	3590.205	30254315	0.2384
50	2300.577	3287805	0	**122.529**	2858813	0	2559.374	17347043	0
100	3600.000	1743982	0.0501	**3600.000**	45793886	0.0251	**3600.000**	10626992	0.0251
100	3600.000	2541105	0.1257	**422.723**	6257926	0	3600.000	5811095	0.1006
100	3600.000	2158927	0.0755	**1527.458**	19645302	0	3600.000	8714045	0.0504
100	3600.000	1667746	0.0503	**3600.000**	26110318	0.0252	3600.000	11935592	0.0503
100	3600.000	3076019	0.0758	**515.755**	2779656	0	3600.000	9821584	0.0506

Table 8.8: Instances *Rand* with $d = 5$

Table 8.9 illustrates the difficulty of instances *Rand* with $d = 10$. Only one instance with 30 nodes has been solved to optimality. Although the algorithms are timed out after

one hour for most instances, Algorithm 37 ends with smaller gaps than the two other strategies.

$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Alg37}	NN_{Alg37}	Gap_{Alg37}	RT_{MTZ}	NN_{MTZ}	Gap_{MTZ}
30	3600.000	6487976	1.4218	**3600.000**	67817193	0.3167	3600.000	28001423	0.5542
30	3600.000	5173073	1.5813	**233.168**	4689614	0	1428.273	5334697	0
30	3600.000	6083729	2.6758	**3600.000**	42766175	1.3097	3600.000	23738907	1.6910
30	3600.000	5855849	1.3077	**3600.000**	72920013	0.6154	3600.000	24333012	0.9231
30	3600.000	6157800	1.6667	**3600.000**	64764523	0.6309	3600.000	20324634	1.1820

Table 8.9: Instances *Rand* with $d = 10$

Similarly with the *Grid* and *Geom* topologies, Tables 8.7, 8.8 and 8.9 show that increasing the number of budget constraints makes *Rand* instances more difficult to solve. We manage to solve instances up to 500 vertices with $d = 2$, 100 nodes with $d = 5$, while we solve instances with 30 vertices with $d = 10$.

The following Tables 8.10, 8.11 and 8.12 report the average separation time (ST), number of separated inequalities (NC) and number of nodes (NN) in the branching tree for the subtour formulation and Algorithm 37. Note that for Algorithm 37, we divide the separated inequalities in two categories; $NC(ST)$ related to spanning tree inequalities, separated by Algorithms 32, 33 and 34, and $NC(C)$ associated with the cover inequalities separated by Algorithm 36. We also give the average number of nodes for the three algorithms.

d	$\|V\|$	ST_{ST}	NC_{ST}	NN_{ST}	ST_{Alg37}	$NC_{Alg37}(ST)$	$NC_{Alg37}(C)$	NN_{Alg37}	NN_{MTZ}
2	100	0.76	866	662	0.47	100	330	952	76631
2	500	14.54	6001	2592	6.26	347	1208	2123	1150407
2	1000	53.68	17012	3839	18.24	681	1152	2189	2068852
2	1500	527.18	69709	1545	29.74	1193	2940	2738	960478
2	2000	936.96	88487	1028	133.05	1306	3074	5663	277704
2	3000	1734.61	90150	3030	262.33	3045	6685	6900	141829
2	4000	1734.85	65843	3001	655.98	4906	7127	11125	74823
5	100	3.36	3011	144512	8.67	132	655	94384	531496
5	200	13.11	7033	1174915	34.78	292	1292	1444908	4466101
5	300	21.66	11533	1275523	65.75	159	1910	1936366	5245169
5	400	30.13	14856	4837015	95.62	511	2568	17170212	3168258
5	500	45.15	18512	1042966	168.42	403	3222	2389149	2595528
5	600	55.84	22439	684258	243.27	792	3917	10505240	1761380
10	100	4.75	3399	2864295	201.03	108	805	19541444	5372252
10	200	13.73	8752	1025241	765.40	163	1320	9628287	4888444

Table 8.10: Statistics on instances *Grid*

d	$\|V\|$	ST_{ST}	NC_{ST}	NN_{ST}	ST_{Alg37}	$NC_{Alg37}(ST)$	$NC_{Alg37}(C)$	NN_{Alg37}	NN_{MTZ}
2	100	0.56	519	1406	0.52	113	591	1479	4485057
2	500	14.77	9487	2051	7.79	1857	2148	1538	2513544
2	1000	165.90	40945	1647	37.77	3145	1629	2780	1057729
2	1500	349.02	128572	1201	102.97	2587	985	6307	347562
2	2000	584.31	77580	1586	257.70	6194	3725	13697	212763
2	2500	746.95	624810	5049	314.76	7248	2007	9339	103248
2	3000	780.62	55483	4016	324.86	13380	2609	8406	61348
5	100	3.03	3132	1509832	10.08	104	642	821392	8237607
5	200	9.24	7243	1530419	40.69	154	1309	6989206	5808224
5	300	10.68	6687	1403744	228.37	403	1957	10017927	4463209
10	50	3.31	1393	3917606	63.38	104	326	5836971	12716714
10	100	5.88	3107	2082105	226.56	163	657	17184527	11973261

Table 8.11: Statistics on instances *Geom*

d	$\|V\|$	ST_{ST}	NC_{ST}	NN_{ST}	ST_{Alg37}	$NC_{Alg37}(ST)$	$NC_{Alg37}(C)$	NN_{Alg37}	NN_{MTZ}
2	100	1.95	2049	1237	1.77	103	208	661	4351
2	300	46.48	12640	23423	105.74	10585	868	8973	4537
2	500	246.20	4740	4147	125.59	7033	0	2126	1
5	50	8.39	1518	4727536	8.94	104	325	13767447	1691046
5	100	5.13	3430	2155418	22.25	275	641	20117418	8093295
10	30	3.90	824	6494380	47.81	42	200	50591504	18986751

Table 8.12: Statistics on instances *Rand*

The results show that the number of nodes in the branching tree of the subtour formulation and Algorithm 37 explodes in the $d = 5, 10$ cases, contrarily to the $d = 2$ case. Recall that for the three topologies, we manage to solve higher instances for the smallest value of d.

Moreover, in the $d = 2$ case the subtour formulation has a higher total separation time than Algorithm 37, but when $d = 5, 10$ we have the opposite as we run a procedure for every budget constraint in Algorithm 36.

Contrarily to the $d = 1$ case where most separated inequalities in Algorithm 28 are spanning tree's inequalities, the majority of separated inequalities in Algorithm 37 are inequalities in the intersection of both the spanning tree and the multi-knapsack problems, especially for the higher values of d.

Tables 8.13, 8.14 and 8.15 give the resolution time of Algorithm 31 and the quality of the computed solution $T^* = (V, F^*)$. The latter is illustrated by the ratio $\frac{\sum_{e \in F^*} c_e}{\sum_{e \in F'} c_e} - 1$ where $T' = (V, F')$ is an optimal solution of the related MdBSTP. In the case where no feasible solution has been found, we denote $+\infty$.

d	$\|V\|$	Time (s)	Ratio (%)	d	$\|V\|$	Time (s)	Ratio (%)
2	500	0.097	0.0009	2	4000	1.402	0.0002
2	500	0.071	0.1456	2	4000	1.172	0.0020
2	500	0.069	0.1687	2	4000	0.896	0.0005
2	500	0.067	0.0275	2	4000	0.958	0.0002
2	1000	0.153	0.0022	5	500	0.132	0.0103
2	1000	0.156	0.0009	5	500	0.111	0.0448
2	1000	0.151	0.0007	5	500	0.104	0.0405
2	1000	0.161	0.1612	5	500	0.113	0.0558
2	1500	0.238	0.1708	5	1000	5.365	0.0123
2	1500	0.246	0.0030	5	1000	0.253	0.0346
2	1500	0.240	0.0003	5	1000	0.218	0.0381
2	1500	0.274	0.0009	5	1000	0.218	0.0364
2	2000	0.331	0.1805	10	100	0.028	0.2491
2	2000	0.325	0.1581	10	100	0.034	0.1566
2	2000	0.331	0.1664	10	100	0.029	0.2303
2	2000	0.340	0.0006	10	100	0.032	0.2144
2	3000	0.786	0.1715				
2	3000	0.588	0.0011				
2	3000	0.682	0.0011				
2	3000	0.594	0.0007				

Table 8.13: Statistics on instances *Grid*

d	$\|V\|$	Time (s)	Ratio (%)	d	$\|V\|$	Time (s)	Ratio (%)
2	500	0.173	0.0040	2	3000	2.142	0.0169
2	500	0.151	0.0252	2	3000	1.860	0.0097
2	500	0.146	0.0122	2	3000	2.028	0.0010
2	500	0.139	0.0052	2	3000	1.920	0.0022
2	1000	0.533	0.0067	5	100	0.033	0.1873
2	1000	0.336	0.0016	5	100	0.053	0.0239
2	1000	0.319	0.0017	5	100	0.036	0.1809
2	1000	0.336	0.0005	5	100	0.045	0.2322
2	1500	0.643	0.0008	5	500	0.306	0.0678
2	1500	0.638	0.0057	5	500	0.253	0.0606
2	1500	0.702	0.0086	5	500	0.976	0.0261
2	1500	0.552	0.0089	5	500	0.224	0.0584
2	2000	1.043	0.0109	10	50	0.036	0.0835
2	2000	1.084	0.0013	10	50	0.061	$+\infty$
2	2000	1.312	0.0048	10	50	0.033	0.1221
2	2000	0.968	0.0142	10	50	0.033	$+\infty$

Table 8.14: Statistics on instances *Geom*

d	$\|V\|$	Time (s)	Ratio (%)	d	$\|V\|$	Time (s)	Ratio (%)
2	500	11.803	0.1349	5	100	0.359	0.0186
2	500	10.697	0.1396	5	100	0.442	0.0219
2	500	11.976	0.0571	5	100	0.399	0.0587
2	500	10.935	0.1358	5	100	0.394	0.0554
2	1000	48.778	0.3338	10	30	0.051	0.2617
2	1000	49.989	0.3029	10	30	0.047	0.2635
2	1000	51.424	0.3079	10	30	0.043	0.2509
2	1000	48.896	0.3687	10	30	0.047	0.2773

Table 8.15: Statistics on instances *Rand*

The results show that Algorithm 31 is very fast, especially in instances *Grid* and *Geom*. Moreover, the ratios illustrate the good quality of the feasible solution found by the heuristic algorithm. Although those solutions are close to an optimal solution, using them as MIP Start as in the $d = 1$ case is not efficient in practice. Notice that they can be used as MIP Start for instances *Rand* with $d = 2$ as no feasible solutions are found in one hour on instances with 500 nodes.

Chapter 9

Conclusion and future works

In this thesis, we have proposed an exact extended formulation for the 1BSTP based on matroid intersection and disjunctive programming. By carefully selecting rays of the projection cone, we generate valid inequalities that are added to strengthen the linear relaxation. We incorporate the projection technique in a Branch-and-Cut algorithm and computational experiments show that our procedure outperforms state of the art algorithms. The major difficulty to devise strong valid inequalities is to exploit both the spanning tree and the knapsack polytopes. As a future work, it would be interesting to generate more complex rays that yield deeper cuts. The running time to generate these rays and exploit them not only at the root but also during the course of the Branch-and-Cut is an issue. The exponential size of the exact formulation makes it difficult to generate efficiently "good" rays to cut off fractional solutions. An important extension of this work is to devise a good approximation of the convex hull based on a smaller size extended formulation and exploit it to efficiently generate cuts.

Adding more budget constraints makes the problem difficult to solve. We adapt our Branch-and-Cut algorithm to the dBSTP problem and give computational experiments showing that our algorithm outperforms generic ones. Exploiting the conflicts between the knapsack constraints and carefully aggregating them to exploit our techniques for the 1BSTP would be an interesting research issue.

We propose an integer formulation for the MdBSTP when the graph is a cactus. As future work, it would be of interest to extend those results on any graph depending on their treewidth.

Finally, it would be also interesting to consider other network problems subject to one or more knapsack constraints and exploit this relation between a well-structured problem and a volatile problem in order to obtain efficient algorithms.

Chapter 10

Appendices

Appendix 1

Given a spanning tree $T = (V, F)$ and an edge $e \in E \backslash F$, finding the unique cycle created by adding e to T, and swapping e with an edge in this cycle are subroutines of several of our algorithms. By considering a particular structure, we manage to perform swaps very quickly in practice. We construct an arborescence $T' = (V, A)$ related to T, where a sink $s \in V$ of T' is arbitrarily chosen. In order to obtain the cycle created by adding $e = uv$ to T, we consider the directed $s - u$ and $s - v-$ paths in T' and delete all arcs in common in these two paths. The procedure is effective in practice as we may not explore all the graph. Notice that if one performs a swap of uv with an edge in the created cycle, updating the related arborescence can be done by exploring only the arcs that belong to the cycle. Figure 10.1 illustrates this implementation idea.

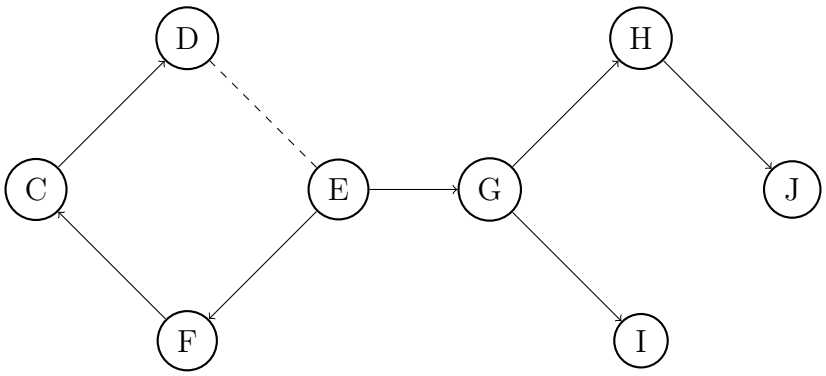


Figure 10.1: Example of an arborescence where E is the sink. The cycle induced by adding the edge DE can be found by considering a small part of the entire graph.

Appendix 2

In Table 10.1, we compare the resolution times of the subtour Algorithm 17 with a Branch-and-Cut based on the cut formulation (3.9)-(3.12) for the M1BSTP. The results show that Algorithm 17 outperforms the cut formulation. The latter times out after one hour for moderate sized instances of the three topologies. The number of nodes in the branching tree of the cut formulation is systematically higher than the one of the subtour formulation.

Instance	$\|V\|$	RT_{ST}	NN_{ST}	Gap_{ST}	RT_{Cut}	NN_{Cut}	Gap_{Cut}
I_{Grid}	250	**0.322**	100	0	36.129	62718	0
I_{Grid}	250	**0.493**	158	0	111.676	224112	0
I_{Grid}	250	**0.920**	450	0	3600.000	3810817	0.0530
I_{Grid}	250	**0.674**	188	0	20.248	8135	0
I_{Grid}	500	**5.070**	485	0	3600.000	764874	0.1911
I_{Grid}	500	**4.867**	297	0	3600.000	684806	0.1918
I_{Grid}	500	**4.669**	415	0	3600.000	673566	$+\infty$
I_{Grid}	500	**5.112**	400	0	3600.000	800491	$+\infty$
I_{Geom}	250	**0.725**	161	0	3600.000	2031016	0.0872
I_{Geom}	250	**1.393**	285	0	3600.000	440125	0.4966
I_{Geom}	250	**1.143**	269	0	3600.000	1621999	0.0384
I_{Geom}	250	**2.097**	195	0	71.484	91054	0
I_{Geom}	500	**20.435**	358	0	3600.000	282660	$+\infty$
I_{Geom}	500	**16.121**	374	0	3600.000	306135	$+\infty$
I_{Geom}	500	**28.264**	456	0	3600.000	298445	$+\infty$
I_{Geom}	500	**14.183**	373	0	3600.000	307859	$+\infty$
I_{Rand}	250	**53.880**	749	0	249.621	12007	0
I_{Rand}	250	**27.731**	1103	0	597.910	20596	0
I_{Rand}	250	**8.151**	319	0	193.522	3719	0
I_{Rand}	250	**33.984**	1270	0	1186.881	7430	0
I_{Rand}	500	3600.000	4303	$+\infty$	3600.000	2522	$+\infty$
I_{Rand}	500	3600.000	3942	$+\infty$	3600.000	2994	$+\infty$
I_{Rand}	500	3600.000	4048	$+\infty$	3600.000	2253	$+\infty$
I_{Rand}	500	**2915.758**	2973	0	3600.000	2816	$+\infty$

Table 10.1: Comparison between the subtour and the cut formulations for the M1BSTP.

Appendix 3

In Table 10.2, we compare two variants of separation algorithm for the subtour inequalities based on [59] for the M1BSTP. The first one, Algorithm 2, adds only the most violated valid inequality at every call and the second one, Algorithm 18, adds every encountered violated inequalities. The results show that adding several violated inequalities performs better in practice.

Instance	$\|V\|$	$RT_{Alg.2}$	$ST_{Alg.2}$	$NN_{Alg.2}$	$RT_{Alg.18}$	$ST_{Alg.18}$	$NN_{Alg.18}$
I_{Grid}	500	21.509	20.15	833	**3.673**	2.78	331
I_{Grid}	500	11.987	10.95	737	**2.887**	1.92	271
I_{Grid}	500	14.979	13.91	739	**1.947**	1.34	218
I_{Grid}	500	14.651	13.18	729	**5.256**	3.38	300
I_{Grid}	1500	2697.437	1450.93	6574	**1110.847**	304.35	1150
I_{Grid}	1500	1387.073	1049.58	5926	**204.007**	61.44	764
I_{Grid}	1500	3276.304	2225.72	6341	**827.897**	190.46	731
I_{Grid}	1500	2080.821	1662.21	5226	**357.843**	102.09	733
I_{Geom}	500	34.186	30.40	1047	**19.281**	8.17	509
I_{Geom}	500	88.974	65.58	1151	**35.667**	11.17	587
I_{Geom}	500	20.945	17.10	815	**9.374**	4.18	316
I_{Geom}	500	37.724	30.71	1026	**14.317**	4.42	471
I_{Geom}	1000	3600.000	924.61	9511	**385.189**	52.90	907
I_{Geom}	1000	1664.293	839.34	6184	**303.118**	48.04	685
I_{Geom}	1000	2239.241	1335.02	5988	**240.958**	51.34	743
I_{Geom}	1000	3600.000	1530.67	9573	**535.795**	77.95	773
I_{Rand}	300	380.760	52.92	3514	**295.021**	39.81	1953
I_{Rand}	300	**138.916**	24.58	1746	167.175	24.87	1952
I_{Rand}	300	**66.828**	17.40	1294	91.348	18.98	1430
I_{Rand}	300	145.625	27.72	2095	**136.662**	28.30	2362
I_{Rand}	500	3494.806	354.54	5673	**2445.293**	291.16	3917
I_{Rand}	500	**1736.621**	212.40	3596	3600.000	230.32	2970
I_{Rand}	500	3600.000	296.21	5112	**3531.116**	244.39	3363
I_{Rand}	500	3600.000	335.64	5222	**1882.832**	260.04	4862

Table 10.2: Comparison between subtour separation algorithms for the M1BSTP.

Appendix 4

In Table 10.3, we compare two upper bound values of K, $20|V|$ and $40|V|$, on the number of separated inequalities added to the model by Algorithm 18 for the M1BSTP. The results show that setting $K = 40|V|$ performs better in practice. By setting $K = 20|V|$, the algorithm (denoted by $ST20$) does not even manage to find a feasible solution in one hour for several huge instances.

Instance	$\|V\|$	RT_{ST20}	NC_{ST20}	NN_{ST20}	Gap_{ST20}	RT_{ST}	NC_{ST}	NN_{ST}	Gap_{ST}
I_{Grid}	1000	132.072	16967	519	0	**130.145**	16967	519	0
I_{Grid}	1000	226.919	20103	1370	0	**220.399**	23645	818	0
I_{Grid}	1000	87.769	13821	565	0	**86.285**	13821	565	0
I_{Grid}	1000	275.189	20121	1172	0	**210.428**	21581	560	0
I_{Grid}	1500	295.156	23608	678	0	**295.106**	23608	678	0
I_{Grid}	1500	678.689	30107	2071	0	**604.989**	34827	827	0
I_{Grid}	1500	2219.563	30130	17254	0	**838.782**	37220	767	0
I_{Grid}	1500	382.872	28267	578	0	**382.121**	28267	578	0
I_{Grid}	2000	**1179.109**	40127	1447	0	2074.972	51695	1146	0
I_{Grid}	2000	3600.000	40703	4775	$+\infty$	**2643.512**	66836	925	0
I_{Grid}	2000	3600.000	40233	3700	$+\infty$	**3600.000**	79089	1284	0.2258
I_{Grid}	2000	3600.000	40485	2950	$+\infty$	**3600.000**	80017	796	0.2383
I_{Geom}	1000	313.655	20018	1152	0	**220.351**	20273	1061	0
I_{Geom}	1000	272.188	20031	662	0	**271.754**	20379	610	0
I_{Geom}	1000	312.692	18986	822	0	**312.050**	189.86	822	0
I_{Geom}	1000	711.487	20102	2946	0	**374.445**	21932	719	0
I_{Geom}	1500	3600.000	30423	5254	$+\infty$	**3600.000**	54762	1074	0.1758
I_{Geom}	1500	3600.000	30444	5159	$+\infty$	**1167.476**	44937	1023	0
I_{Geom}	1500	3600.000	30078	4821	$+\infty$	**1852.831**	40777	1023	0
I_{Geom}	1500	3600.000	30510	5169	$+\infty$	**3472.961**	50664	920	0
I_{Rand}	500	1891.304	9979	2893	0	**1889.523**	9978	2893	0
I_{Rand}	500	3600.000	1035	5337	$+\infty$	**3600.000**	19929	3027	0.0215
I_{Rand}	500	3600.000	9887	5101	$+\infty$	**3600.000**	13858	3234	0.1076
I_{Rand}	500	3600.000	10001	5176	$+\infty$	3600.000	19997	3603	$+\infty$

Table 10.3: Comparison of the subtour formulation with $K = 20|V|$ and $K = 40|V|$ for the M1BSTP.

Appendix 5

In the following, we compare several MTZ based formulations to obtain the best compact competitor for our M1BSTP algorithm. The MTZ formulation, that will be denoted by M, is given by:

$$\min \sum_{ij \in A} c_{i,j} x_{ij} \tag{10.1}$$

$$s.t. \sum_{i \in \delta^-(j)} x_{ij} = 1, \qquad \forall j \in V \backslash \{r\}, \tag{10.2}$$

$$u_i - u_j + |V| x_{ij} \leq |V| - 1, \qquad \forall (i,j) \in A, j \neq r, \tag{10.3}$$

$$u_i \leq |V| - 1, \qquad \forall i \in V \backslash \{r\}, \tag{10.4}$$

$$u_i \geq 1, \qquad \forall i \in V \backslash \{r\}, \tag{10.5}$$

$$u_r = 0, \tag{10.6}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in V, j \in V \backslash \{i,r\}. \tag{10.7}$$

We denote by Var the integer program (10.1)-(10.7) enhanced with the families of inequalities

$$x_{ij} + x_{ji} \leq 1, \qquad ij \in E. \tag{10.8}$$

In Table 10.4, we compare the resolution times of these two MTZ formulations with the enhanced MTZ based formulation given by Agra et al. [2]. The results of the three Branch-and-Bound algorithms do not show a dominance between the three formulations. For the computational experiments, we choose the formulation Var as it performs better in average for instances I_{Rand}, the only instances where MTZ based formulation are competitive. Moreover, this formulation can be generalized to the general case with $d > 1$, contrarily to Agra et al. formulation.

Instance	$\|V\|$	RT_{Var}	NN_{Var}	RT_{M}	NN_{M}	$RT_{[2]}$	$NN_{[2]}$
I_{Grid}	300	124.388	472039	50.551	112826	**30.683**	65187
I_{Grid}	300	18.279	42389	**17.574**	36077	19.891	49890
I_{Grid}	300	11.104	27152	**6.309**	7453	9.743	22498
I_{Grid}	300	**12.065**	24178	13.314	21636	19.756	38374
I_{Grid}	500	**256.621**	340151	1090.573	1662942	548.378	811227
I_{Grid}	500	1128.512	2853839	**464.608**	808276	778.496	1224907
I_{Grid}	500	3600.000	4433367	3600.000	7308538	**3600.000**	3494114
I_{Grid}	500	**3600.000**	3336126	**3600.000**	3554985	**3600.000**	3707062
I_{Geom}	300	**42.532**	4536	49.168	46472	89.315	62186
I_{Geom}	300	18.779	162280	27.493	22303	**15.702**	16042
I_{Geom}	300	832.484	1101208	**301.668**	474301	722.120	1257269
I_{Geom}	300	**216.158**	331137	899.005	1396197	367.067	696618
I_{Geom}	500	3600.000	2737730	**3600.000**	2962800	3600.000	4244664
I_{Geom}	500	**3600.000**	1822726	3600.000	2347717	3600.000	3332630
I_{Geom}	500	2083.081	3151131	2830.673	1922339	**1611.430**	1471475
I_{Geom}	500	**1692.369**	1030807	2405.157	1288167	3600.000	2564085
I_{Rand}	500	10.671	1	20.513	1	**8.057**	1
I_{Rand}	500	**23.867**	1733	41.253	5917	27.618	828
I_{Rand}	500	23.646	10474	31.726	15575	**13.575**	828
I_{Rand}	500	**3600.000**	4359060	**3600.000**	8140321	**3600.000**	9034020
I_{Rand}	1000	**104.527**	1	204.294	4587	144.143	1
I_{Rand}	1000	**72.128**	1	178.389	1	170.806	993
I_{Rand}	1000	**75.723**	1	183.351	1	214.734	1
I_{Rand}	1000	74.371	1	142.358	1	**68.645**	1
I_{Rand}	2000	**742.588**	1	3600.000	1	1585.007	4476
I_{Rand}	2000	397.184	1	1706.636	1	**325.935**	1
I_{Rand}	2000	843.369	1	2117.899	1	**751.484**	1
I_{Rand}	2000	3600.000	9205	**2722.037**	1	3600.000	7858

Table 10.4: Comparison between MTZ formulations M, Var and [2] for the M1BSTP

Bibliography

[1] Vijay Aggarwal, Yash P Aneja, and KPK Nair. Minimal spanning tree subject to a side constraint. *Computers & Operations Research*, 9(4):287–296, 1982.

[2] Agostinho Agra, Adelaide Cerveira, Cristina Requejo, and Eulália Santos. On the weight-constrained minimum spanning tree problem. In *International Conference on Network Optimization*, pages 156–161. Springer, 2011.

[3] Agostinho Agra, Cristina Requejo, and Eulália Santos. Implicit cover inequalities. *Journal of Combinatorial Optimization*, 31:1111–1129, 2016.

[4] Lígia Amado and Paulo Barcia. Matroidal relaxations for 0–1 knapsack problems. *Operations Research Letters*, 14(3):147–152, 1993.

[5] Kim Allan Andersen, Kurt Jörnsten, and Mikael Lind. On bicriterion minimal spanning trees: an approximation. *Computers & Operations Research*, 23(12):1171–1182, 1996.

[6] Rafael Andrade, Abilio Lucena, and Nelson Maculan. Using lagrangian dual information to generate degree constrained spanning trees. *Discrete Applied Mathematics*, 154(5):703–717, 2006.

[7] David L Applegate. *The Traveling Salesman Problem: A Computational Study*, volume 17. Princeton university press, 2006.

[8] Egon Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[9] Francisco Barahona. Separating from the dominant of the spanning tree polytope. *Operations Research Letters*, 12(4):201–203, 1992.

[10] André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128:355–372, 2011.

[11] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to Linear Optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[12] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[13] Stephen Boyd, Lin Xiao, and Almir Mutapcic. Subgradient methods. *lecture notes of EE392o, Stanford University, Autumn Quarter*, 2004(01), 2003.

[14] J Orestes Cerdeira and Paulo Barcia. When is a 0-1 knapsack a matroid? *Portugaliae Mathematica*, 52(4):475–480, 1995.

[15] Altannar Chinchuluun and Panos M Pardalos. A survey of recent developments in multiobjective optimization. *Annals of Operations Research*, 154(1):29–50, 2007.

[16] Gustave Choquet. Étude de certains réseaux de routes. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 206:310–313, 1938.

[17] Wushow Chou and Aaron Kershenbaum. A unified algorithm for designing multidrop teleprocessing networks. In *Proceedings of the third ACM Symposium on Data Communications and Data Networks: Analysis and Design*, pages 148–156, 1973.

[18] Richard Cole, Jeffrey S Salowe, William L. Steiger, and Endre Szemerédi. An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, 18(4):792–810, 1989.

[19] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010.

[20] José Craveirinha, João Clímaco, Lúcia Martins, Carlos G Da Silva, and Nuno Ferreira. A bi-criteria minimum spanning tree routing model for mpls/overlay networks. *Telecommunication Systems*, 52:203–215, 2013.

[21] William H Cunningham. Optimal attack and reinforcement of a network. *Journal of the ACM (JACM)*, 32(3):549–561, 1985.

[22] Ding-Zhu Du and Ker-I Ko. *Theory of Computational Complexity*, volume 58. John Wiley & Sons, 2011.

[23] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[24] Jack Edmonds. Matroids, submodular functions and certain polyhedra. *Combinatorial Structures and Their Applications*, pages 69–87, 1970.

[25] Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1):127–136, 1971.

[26] Peppino Fazio, Floriano De Rango, Cesare Sottile, and Amilcare Francesco Santamaria. Routing optimization in vehicular networks: A new approach based on multiobjective metrics and minimum spanning tree. *International Journal of Distributed Sensor Networks*, 9(11):598675, 2013.

[27] Greg N Frederickson and Roberto Solis-Oba. Increasing the weight of minimum spanning trees. *Journal of Algorithms*, 33(2):244–266, 1999.

[28] Renan Garcia. *Resource Constrained Shortest Paths and Extensions*. Georgia Institute of Technology, 2009.

[29] Mitsuo Gen, Kenichi Ida, and Jongryul Kim. A spanning tree-based genetic algorithm for bicriteria topological network design. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pages 15–20. IEEE, 1998.

[30] Michel X Goemans. Minimum bounded degree spanning trees. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 273–282. IEEE, 2006.

[31] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[32] Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[33] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2. Springer Science & Business Media, 2012.

[34] Zonghao Gu, George L Nemhauser, and Martin WP Savelsbergh. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10(4):427–437, 1998.

[35] Monique Guignard and Moshe B Rosenwein. An application of lagrangean decomposition to the resource-constrained minimum weighted arborescence problem. *Networks*, 20(3):345–359, 1990.

[36] Dan Gusfield. Parametric combinatorial computing and a problem of program module distribution. *Journal of the ACM (JACM)*, 30(3):551–563, 1983.

[37] Refael Hassin and Asaf Levin. An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. *SIAM Journal on Computing*, 33(2):261–268, 2004.

[38] Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[39] Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25, 1971.

[40] Sung-Pil Hong, Sung-Jin Chung, and Bum Hwan Park. A fully polynomial bicriteria approximation scheme for the constrained spanning tree problem. *Operations Research Letters*, 32(3):233–239, 2004.

[41] Mohammad Javadi, Mohamed Lotfi, Gerardo J Osório, Abdelrahman Ashraf, Ali Esmaeel Nezhad, Matthew Gough, and João PS Catalão. A multi-objective model for home energy management system self-scheduling using the epsilon-constraint method. In *2020 IEEE 14th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG)*, volume 1, pages 175–180. IEEE, 2020.

[42] David S Johnson, Maria Minkoff, and Steven Phillips. The prize collecting steiner tree problem: theory and practice. In *SODA*, volume 1, page 4. Citeseer, 2000.

[43] Dylan F Jones and Mehrdad Tamiz. Goal programming in the period 1990–2000. *Multiple Criteria Optimization: State of the art annotated bibliographic surveys*, pages 129–170, 2002.

[44] Hans Kellerer, Ulrich Pferschy, David Pisinger, Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Multidimensional Knapsack Problems*. Springer, 2004.

[45] T Jayanth Kumar and Purusotham Singamsetty. An exact algorithm for multi-constrained minimum spanning tree problem. *International Journal of Mathematics in Operational Research*, 12(3):317–330, 2018.

[46] Harry Loberman and Arnold Weinberger. Formal procedures for connecting terminals with a minimum total wire length. *Journal of the ACM (JACM)*, 4(4):428–437, 1957.

[47] Dinh The Luc. Pareto optimality. *Pareto Optimality, Game Theory and Equilibria*, pages 481–515, 2008.

[48] Thomas L Magnanti and Laurence A Wolsey. Optimal trees. *Handbooks in Operations Research and Management Science*, 7:503–615, 1995.

[49] R Kipp Martin. Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters*, 10(3):119–128, 1991.

[50] R Kipp Martin, Ronald L Rardin, and Brian A Campbell. Polyhedral characterization of discrete dynamic programming. *Operations Research*, 38(1):127–138, 1990.

[51] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM (JACM)*, 30(4):852–865, 1983.

[52] Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical Aspects of Local Search*, volume 25. Springer, 2007.

[53] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[54] Boris Mirkin. *Mathematical Classification and Clustering*, volume 11. Springer Science & Business Media, 2013.

[55] Theodore S Motzkin, Howard Raiffa, Gerald L Thompson, and Robert M Thrall. The double description method. *Contributions to the Theory of Games*, 2(28):51–73, 1953.

[56] Subhash C Narula and Cesar A Ho. Degree-constrained minimum spanning tree. *Computers & Operations Research*, 7(4):239–249, 1980.

[57] Neil Olver and Rico Zenklusen. Chain-constrained spanning trees. *Mathematical Programming*, 167:293–314, 2018.

[58] James G Oxley. *Matroid Theory*, volume 3. Oxford University Press, USA, 2006.

[59] Manfred W Padberg and Laurence A Wolsey. Trees and cuts. In *North-Holland Mathematics Studies*, volume 75, pages 511–517. Elsevier, 1983.

[60] Sandro Pirkwieser, Günther R Raidl, and Jakob Puchinger. Combining lagrangian decomposition with an evolutionary algorithm for the knapsack constrained maximum spanning tree problem. In *Evolutionary Computation in Combinatorial Optimization: 7th European Conference, EvoCOP 2007, Valencia, Spain, April 11-13, 2007. Proceedings 7*, pages 176–187. Springer, 2007.

[61] David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

[62] Franco P Preparata and Michael I Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[63] Anthony Przybylski and Xavier Gandibleux. Multi-objective branch and bound. *European Journal of Operational Research*, 260(3):856–872, 2017.

[64] Rosa M Ramos, Sergio Alonso, Joaquín Sicilia, and Carlos González. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111(3):617–628, 1998.

[65] Ram Ravi and Michel X Goemans. The constrained minimum spanning tree problem. In *Algorithm Theory—SWAT'96: 5th Scandinavian Workshop on Algorithm Theory Reykjavík, Iceland, July 3–5, 1996 Proceedings 5*, pages 66–75. Springer, 1996.

[66] Cristina Requejo and Eulália Santos. Lagrangean based algorithms for the weight-constrained minimum spanning tree problem. In *Agra, Agostinho and Doostmohammadi, Mahdi (2011) A Polyhedral Study of Mixed 0-1 Set. In: Proceedings of the 7th ALIO/EURO Workshop. ALIO-EURO 2011, Porto, pp. 57-59.*, page 38, 2011.

[67] Cristina Requejo and Eulália Santos. Efficient lower and upper bounds for the weight-constrained minimum spanning tree problem using simple lagrangian based algorithms. *Operational Research*, 20(4):2467–2495, 2020.

[68] Phillippe Samer and Sebastián Urrutia. A branch and cut algorithm for minimum spanning trees under conflict constraints. *Optimization Letters*, 9(1):41–55, 2015.

[69] Martin WP Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[70] Alexander Schrijver et al. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24. Springer, 2003.

[71] Andrew W Shogan. Constructing a minimal-cost spanning tree subject to resource constraints and flow requirements. *Networks*, 13(2):169–190, 1983.

[72] Francis Sourd and Olivier Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3):472–484, 2008.

[73] Thomas Stidsen, Kim Allan Andersen, and Bernd Dammann. A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science*, 60(4):1009–1032, 2014.

[74] Robert Endre Tarjan. *Data Structures and Network Algorithms*. SIAM, 1983.

[75] E.L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.

[76] Gary R Waissi. Network flows: Theory, algorithms, and applications, 1994.

[77] Bin Wang and Jennifer C Hou. Multicast routing and its qos extension: problems, algorithms, and protocols. *IEEE network*, 14(1):22–36, 2000.

[78] Laurence A Wolsey. Faces for a linear inequality in 0–1 variables. *Mathematical Programming*, 8(1):165–178, 1975.

[79] Laurence A Wolsey and George L Nemhauser. *Integer and Combinatorial Optimization*. John Wiley & Sons, 2014.

[80] Pan Xu, Aravind Srinivasan, Kanthi K Sarpatwar, and Kun-Lung Wu. Budgeted online assignment in crowdsourcing markets: Theory and practice. In *AAMAS*, pages 1763–1765, 2017.

[81] Ying Xu, Victor Olman, and Dong Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics*, 18(4):536–545, 2002.

[82] Takeo Yamada, Kohtaro Watanabe, and Seiji Kataoka. Algorithms to solve the knapsack constrained maximum spanning tree problem. *International Journal of Computer Mathematics*, 82(1):23–34, 2005.

RÉSUMÉ

Le problème de l'arbre couvrant d-budgété combine les structures de l'arbre couvrant et du sac-à-dos multidimensionnel. On retrouve ce problème dans de nombreuses applications, notamment dans le domaine de la conception de réseaux. Les différentes contraintes de budget permettent de prendre en compte plusieurs critères, potentiellement conflictuels, que l'on retrouve dans de nombreux problèmes réels. Nous considérons des algorithmes exacts, basés sur une approche polyédrale, pour résoudre le problème de l'arbre couvrant budgété avec une ou plusieurs contraintes de budget. Dans le cas où $d = 1$, nous présentons une formulation étendue exacte basée sur des intersections de matroïdes et la programmation disjonctive. En raison de la NP-difficulté du problème, cette formulation est exponentiellement grande et ne peut être résolue ou projetée telle quelle. Nous exploitons donc la structure du cone de projection afin de sélectionner des rayons qui combinent les polytopes de l'arbre couvrant et du sac-à-dos. Nous intégrons cette technique de projection ainsi que plusieurs routines de séparation efficaces dans un algorithme de coupes et de branchements, et les résultats expérimentaux montrent que notre procédure domine les algorithmes de l'état de l'art. Nous généralisons notre algorithme pour résoudre le problème avec $d \geq 2$ contraintes de budget.

MOTS CLÉS

Optimisation combinatoire, Polyèdre, Arbre couvrant, Sac-à-dos, Matroïde, Formulation étendue, Projection, Inégalité valide, Relaxation Lagrangienne, Algorithme de coupes et branchements.

ABSTRACT

The d-Budgeted Spanning Tree Problem lies at the intersection of the spanning tree and the multi-knapsack problems. It models several applications in the fields of network design, power and transportation networks. The multi-knapsack part allows to consider several criteria, potentially in conflict, that occur in many real-world problems. We consider exact algorithms, based on a polyhedral approach, to solve the problem with one or several budget constraints. In the case where d = 1, we give an exact extended formulation based on matroids intersection and disjunctive programming. Due to the NP-hardness of the problem, this formulation is exponentially large and prohibitive to solve. Moreover, there is no hope to obtain all the facets of the convex hull by projection. Instead, we carefully select rays of the projection cone that exploit the spanning tree and the knapsack polytopes. We develop a Branch-and-Cut algorithm based on a careful selection of rays of the projection cone, efficient separation routines, and a fast algorithm to solve the Lagrangian relaxation of the problem. Our experiment results show that our approach outperforms state of the art ones. We generalize our algorithm to solve the problem with $d \geq 2$ budget constraints.

KEYWORDS

Combinatorial optimization, Polyhedra, Spanning tree, Knapsack, Matroid, Extended formulation, Projection, Valid inequality, Lagrangian relaxation, Branch-and-Cut.