N° d'ordre : 25/2017-C/MT

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté de Mathématiques

THÈSE

Présentée pour l'obtention du diplôme de Doctorat $3^{ème}$ cycle (LMD)
En : MATHÉMATIQUES
Spécialité : Recherche Opérationnelle et Management

Par : Asma BOUMESBAH

Sujet

APPROCHE DE RÉSOLUTION DU PROBLÈME DE L'ARBRE MULTIOBJECTIF

Soutenue publiquement le :11/05/2017 à 9h30, devant le jury composé de :

M. A. BERRACHEDI	Professeur	à l'USTHB	Président
M. M.E-A CHERGUI	MCA	à l'USTHB	Directeur de thèse
M. M.O. BIBI	Professeur	à l'UAM.Bejaia	Examinateur
M. M. BOUDHAR	Professeur	à l'USTHB	Examinateur
M. M. MOULAI	Professeur	à l'USTHB	Examinateur
M. M.S. RADJEF	Professeur	à l'UAM.Bejaia	Examinateur
M. A. YASSINE	Professeur	à l'U. HAVRE-France	Examinateur

Remerciments

Louange à Allah, Clément et
Miséricordieux, sans lui rien de tout
cela n'aurait pu être

Je remercie très chaleureusement mon directeur de thèse, Dr CHERGUI Mohamed El-Amine, qui a accepté de prendre la direction de cette thèse, transformant les difficultés rencontrées en une expérience enrichissante. Je lui suis également reconnaissante de m'avoir assuré un encadrement rigoureux tout au long de ces années. Monsieur CHERGUI a su diriger mes travaux avec beaucoup de disponibilité, de tact et d'intérêt, ses qualités scientifiques, pédagogiques et surtout humaines.

Mes vifs remerciements sont également adressés au Professeur A. BERRACHEDI qui a eu l'amabilité de bien vouloir présider le jury. Je remercie très chaleureusement les Professeurs : M.O. BIBI, M. BOUDHAR, M. MOULAI, M.S. RADJEF et A. YASSINE qui ont accepté, sans réserve aucune, d'évaluer cette thèse à sa juste valeur, et de me faire part de leur remarques sûrement pertinentes qui, contribueront, sans nul doute, au perfectionnement du présent travail.

Je remercie spécialement monsieur MEZGHICHE Abdelhak pour ses encouragements, ses conseils et ses appuis, et aussi pour m'avoir accueilli dans son bureau durant toutes ces années. Un grand remerciement à monsieur M. BENDRAOUCHE pour sa disponibilité et pour avoir corrigé mon papier rédigé en anglais.

Je remercie également tous les membres du laboratoire RECITS de l'USTHB pour la bonne humeur et la convivialité qui y régnaient. J'exprime ma profonde gratitude à la responsable de la bibliothèque Taous OULMANE, à Samia GASMI, à tous les employés et à tous mes enseignants.

Je ne saurais terminer sans souligner le soutien amical et chaleureux de mes copines Asma LOUARDANI et Meriem TIACHACHAT de tous les jours qui m'ont soutenu durant ce parcours. Je m'abstiens de les nommer tellement la liste est longue. Je nommerai tout de même mes amies Hadjer BELKHIRI, Hadjer OUZZANE et Wafa KAROUCHE.

Mes remerciements vont particulièrement à mes très cher parents, qui m'ont constamment encouragés et soutenus tout au long de ces années et qui ont toujours été à mes côtés. Je ne saurai passer sous silence l'apport inestimable des autres membres de ma famille, Souad, mes sœurs et leurs époux, frères et leurs épouses, mon cher oncle et père Abbas et à sa petite famille, aussi à mes nièces, neveux, cousines et cousins qui m'ont soutenus, de près ou de loin durant mes études.

Je remercie Allah de m'avoir entouré de personnes formidables qui ont contribué, chacune à sa façon, et ce, à différentes étapes de mon cheminement, d'une manière ou d'une autre, à la réalisation de cette thèse de doctorat.

Table des matières

Introduction **1**

1 Concepts de graphes **4**

1.1 Généralités sur les graphes 5

1.1.1 Notions et concepts fondamentaux sur les graphes 5

1.1.2 Arbres 8

1.2 Problème de l'arbre de poids minimum 11

1.2.1 Description du problème 11

1.2.2 Algorithmes de résolution du problème 12

1.2.3 Applications du problème 13

2 Programmation linéaire **15**

2.1 Résultats fondamentaux de la programmation linéaire 15

2.1.1 Formes matricielles classiques 17

2.1.2 Méthode de résolution d'un programme linéaire 19

2.2 Programmation linéaire en nombres entiers (PLNE) 21

2.2.1 Quelques méthodes de résolution des PLNE 22

2.2.2 "Branch and Bound" 22

2.2.3 Coupes de Gomory 25

3 Programmation linéaire multiobjectif **27**

3.1 Sur la programmation multiobjectif linéaire discrète 28

3.1.1 Terminologie - Définitions 29

3.1.2 Notion de cônes 31

3.2 Détection graphique de l'efficacité 33
3.2.1 Solutions supportées et solutions non supportées 34
3.2.2 Fonctions scalarisantes 34
3.2.3 Approches de résolution 41
3.2.4 Quelques méthodes exactes de résolution 42

4 Problème de l'arbre multiobjectif (MOST) - État de l'art 49
4.1 Position du problème 49
4.1.1 Définitions et notations 49
4.1.2 Base de cycles 50
4.1.3 Modèle mathématique 51
4.2 État de l'art 52
4.2.1 Extension multicritère des algorithmes "Prim" et "Kruskal" 52
4.2.2 Méthodes d'optimisation combinatoire multiobjectif appliquées au problème MOST 55

5 Méthode exacte pour la résolution du problème de l'arbre multiobjectif 60
5.1 Une méthode exacte de résolution de MOST 60
5.1.1 Principe de la méthode exacte 61
5.2 Procédures de l'algorithme 63
5.3 Principaux résultats 66
5.4 Résultats expérimentaux 71

6 Une méthode approchée GA-VNS pour MOST 75
6.1 Principe de la méthode approchée GA-VNS 80
6.1.1 Adaptation de l'algorithme GA-VNS 80
6.2 Résultats expérimentaux 86

Conclusion et perspectives 88

Bibliographie 90

Table des figures

1.1 Graphe simple et non orienté G 9
1.2 Arbre de G 9
1.3 Graphe pondéré G 11
1.4 Arbre de coût minimum de G 12

2.1 Exemple d'arbre de recherche 23
2.2 Exemple d'arbre binaire 23

3.1 Représentation du point idéal et du point "nadir". 31
3.2 Exemples d'ensemble convexe et d'ensemble non convexe. 31
3.3 Schéma des différentes solutions et points caractéristiques de l'espace des objectifs 36
3.4 Ensemble réalisable du problème ($MOIPL1$) dans l'espace des décisions et l'espace des objectifs de l'exemple 3.1 38
3.5 Ensemble réalisable du problème ($MOIPL1$) dans l'espace des décisions et l'espace des objectifs de l'exemple 3.2 39
3.6 Solutions efficaces supportées et non supportées 40
3.7 Régions des solutions efficaces non supportées 41
3.8 Illustration de la méthode $\epsilon-$contrainte. 45

4.1 Contre-exemple pour la version modifiée de l'algorithme de Corley (1985) [17] 54

5.1 Un graphe connexe G 69
5.2 Un graphe connexe G traité dans [2] 71

6.1 Principe de l'algorithme NSGA-II 78

6.2 Distance de Crowding 78
6.3 Recherche à voisinage variable (Hansen et *al.*, 1997) 80
6.4 Croisement à deux points 83

Liste des tableaux

2.1 Écriture canonique de $(P)/B$ 20

5.1 Résultats expérimentaux 72

5.2 Résultats en moyenne pour les graphes ayant un petit nombre d'arêtes de type $e2c$ 74

6.1 Comparaison de GA-VNS avec un croisement à un point et celui à deux points 82

6.2 Comparaison de GA-VNS avec les procédures 2 et 3 84

6.3 Comparaison entre $MOST$-Algorithme et GA-VNS 86

6.4 Comparaison entre GA-VNS et KEA 87

Introduction

L'optimisation combinatoire est un domaine intensivement étudié en raison de son potentiel d'application à de nombreux problèmes rencontrés dans des situations réelles. Les recherches dans ce domaine sont longtemps restées confinées aux situations où un unique objectif est à optimiser. Cependant, de nombreux problèmes d'optimisation pratique impliquent généralement la minimisation (ou la maximisation) de plusieurs critères de décision contradictoires. Par exemple, dans le problème de conception de réseau topologique, il est souhaitable de trouver la meilleure disposition des composants optimisant les critères de performance, tels que le coût financier, le délai de message, le trafic, la fiabilité de liaison, etc. Ces critères sont contradictoires et ne peuvent pas être optimisés simultanément. Il faut plutôt trouver un compromis satisfaisant. Ainsi, un décideur doit choisir la meilleure solution de compromis, en tenant compte de la préférence des critères. Le but de l'optimisation combinatoire multicritères (MCCO) est d'optimiser simultanément r critères ou r objectifs, $r > 1$ et de trouver un compromis satisfaisant. Les problèmes MCCO ont un ensemble de solutions optimales (au lieu d'un seul optimum) en ce sens qu'aucune autre solution ne leur est supérieure lorsque tous les critères sont pris en compte. Elles sont connues comme Pareto optimales ou solutions efficaces.

La résolution des problèmes de MCCO est très différente de celle du cas monobjectif ($r = 1$), où une solution optimale est recherchée. La difficulté n'est pas seulement due à la complexité combinatoire comme dans le cas d'un seul objectif, mais aussi à la recherche de tous les éléments de l'ensemble efficace, dont la cardinalité croît avec le nombre d'objectifs. Dans la littérature, certains auteurs ont proposé des méthodes exactes pour résoudre des problèmes spécifiques aux problèmes de MCCO [26]. Ces méthodes sont généralement valables pour des problèmes biobjectif ($r = 2$) mais ne peuvent pas être facilement adaptées

à un plus grand nombre d'objectifs. En outre, les méthodes exactes sont inefficaces pour résoudre à grande échelle des problèmes de MCCO NP-difficiles. Comme dans le cas d'un seul critère, l'utilisation de techniques heuristiques et métaheuristiques semble être l'approche la plus prometteuse pour MCCO en raison de leur efficacité, de leur généralité et de leur simplicité relative de mise en œuvre. Ces techniques génèrent de bonnes solutions approximatives de bon compromis dans un temps de calcul raisonnable.

Le problème de l'arbre minimum multiobjectif ($MOST$), est une représentation plus réaliste des problèmes pratiques dans le monde réel et se pose dans beaucoup de contextes. Comme exemple, nous citons celui de l'optimisation de la qualité de service dans un réseau de télécommunication dont l'évaluation est exprimée selon un ensemble de critères : délai minimum, débit maximum, taux d'erreurs minimum, coût d'utilisation minimum, etc. Ce problème est connu pour être NP-difficile même pour $r = 2$ [10, 40] et à notre connaissance, peu d'approches de résolution valides ont été développées pour le cas biobjectif, sont théoriquement généralisées pour des problèmes comportant plus de deux objectifs [14, 40], sans avoir effectué une étude expérimentale.

Dans ce présent travail, nous proposons une méthode exacte pour trouver l'ensemble complet efficace sans aucune restriction sur le nombre de critères considérés. Son principe de recherche arborescente repose sur un procédé en deux étapes exécutées en alternance, la séparation d'abord par rapport à des arêtes communes à au moins deux cycles du graphe donné, induisant l'étape de construction des contraintes du problème sous forme d'égalités et d'inégalités linéaires qui assurent la non création des cycles. Chacune des branches de l'arborescence induit un programme multiobjectif linéaire discret en variables bivalentes ($MOLBP$) modélisant le problème correspondant à cette branche. Par conséquent, les programmes $MOLBP$ obtenus peuvent être résolus par l'une des deux méthodes décrites dans [1] et [51].

On présente aussi dans cette thèse une méthode approchée qui permet de trouver une meilleure approximation du front de Pareto pour le problème $MOST$. L'approche correspond à une hybridation de deux métaheuristiques : "Non-dominates Sorting Genetic" Algorithm (NSGA-II) [23] et "Variable Neighborhood Search" (VNS) [42]. On commence par l'adaptation de NSGA-II sur notre problème en générant une population initiale qui contient des arbres efficaces supportés ainsi, un nouvel opérateur de croisement à deux

points et une heuristique de mutation sont appliqués pour obtenir la population fils et finalement les éléments du premier front obtenu dans l'opérateur de sélection sont améliorés par l'algorithme VNS. Cette approche consiste en l'avantage des algorithmes NSGA-II et VNS et permet de trouver de bons individus pour contrebalancer entre la diversification et l'intensification au cours du processus de recherche.

L'organisation de cette thèse est la suivante :

Nous présentons des notions essentielles de la théorie des graphes utilisées dans toute la suite du document dans le chapitre 1. Le chapitre 2, a pour objectif de présenter le contexte et les résultats concernant la programmation linéaire, notamment les méthodes de résolution classiques (méthodes du simplexe, coupes de Gomory et "Branch & Bound"). Ensuite, dans le chapitre 3, l'essentiel des définitions et résultats liés à l'optimisation multiobjectif en nombres entiers est présenté et quelques méthodes d'optimisation multiobjectif discrète existantes dans la littérature sont relatées. Le chapitre 4 présente le problème de l'arbre multiobjectif et des méthodes exactes dédiées à ce problèmes pour le cas biobjectif. Notre contribution par une méthode exacte et une méthode approchée pour la résolution du problème posé sont exposées en détails dans les chapitres 5 et 6 respectivement. On finalise chacune des deux méthodes par une étude expérimentale réalisée sous environnement Matlab 2014a, sur PC portable *h*p, core i5, 8 Go de rame, sur des instances générées aléatoirement.

Chapitre 1

Concepts de graphes

La théorie des graphes est, avec la combinatoire, une des pierres angulaires de ce qu'il est commun de désigner par mathématiques discrètes. En effet, la théorie des graphes apparue dans le magasin des curiosités mathématiques "les ponts de Königsberg", puis devenue un outil pour l'étude des circuits électriques "Kirshhoff", elle a été utilisée par la chimie, la psychosociologie et l'économie avant même d'avoir été constituée. Elle constitue une branche à part entière des mathématiques, grâce aux travaux de König, Menger, Cayley puis de Berge et d'Erdös [4].

Les applications de la théorie des graphes et de la recherche opérationnelle sont aujourd'hui immenses : aide à la décision, stratégie, optimisation (plus court chemin, arbre de coût minimum), réseaux de transports : chemins de fer, lignes aériennes, électricité, gaz, oléoducs (transport de l'énergie), internet (réseau de l'information), ports et aéroports, ordonnancement des tâches, etc.

Un graphe peut représenter toutes sortes de situations dans les phénomènes d'organisation, par exemple, un réseau, c'est à dire un graphe comportant une entrée et une sortie, peut correspondre à des canalisations où circule un fluide (liquide, gaz). Les graphes constituent donc une méthode de pensée qui permet la représentation (modélisation) et la résolution de problèmes discrets que pose la recherche opérationnelle.

1.1 Généralités sur les graphes

1.1.1 Notions et concepts fondamentaux sur les graphes

Les différentes définitions et notions données dans cette section sont prises des livres suivants : [4, 5, 11, 31, 57].

I. Graphe orienté

Un **graphe orienté** ou digraphe (directed graph) $G = (V, E)$ est défini par l'ensemble fini $V = \{v_1, v_2, ..., v_n\}$ dont les éléments sont appelés sommets (Vertices en anglais), et par l'ensemble fini $E = \{e_1, e_2, ..., e_m\}$ dont les éléments sont appelés arcs (Edges en anglais).
- Pour un **arc** $e = (u, v)$ de l'ensemble E, on dit que $u \in V$ est l'extrémité initiale et $v \in V$ l'extrémité finale de e.

II. Graphe non orienté

Un **graphe non orienté** $G = (V, E)$ est défini par l'ensemble fini $V = \{v_1, v_2, ..., v_n\}$ dont les éléments sont appelés sommets, et par l'ensemble fini $E = \{e_1, e_2, ..., e_m\}$ dont les éléments sont appelés arêtes (edges en anglais).
- Une **arête** e de l'ensemble E est définie par une paire non ordonnée de sommets, appelés les extrémités de e.
- Si l'arête e relie les sommets u et v, on dira que ces sommets sont **adjacents**, ou **incidents** avec e, ou bien que l'arête e est incidente avec les sommets u et v.

Dans tout ce qui suit, on considère des graphes non orientés.
- Une arête dont les extrémités sont identiques est appelée **une boucle**, et une arête dont les extrémités sont distinctes **un lien**. Deux liens ou plus ayant la même paire d'extrémités sont appelés des arêtes **parallèles**.
- Un graphe est **simple** s'il n'a ni boucle ni arêtes parallèles.
- On appelle **ordre** (Order) d'un graphe le nombre de ses sommets.
- Le **degré** (Degree) d'un sommet est le nombre d'arêtes dont il est extrémité.
- Le **voisinage** (neighborhood) d'un sommet est l'ensemble de tous ses sommets adjacents.

III. Matrices d'un graphe

Définition 1.1. Une matrice d'adjacences (adjacency matrix) d'un graphe non orienté simple $G = (V, E)$ est une matrice carrée contenant des 0 et des 1, dont les lignes et les colonnes sont classées par sommets. Un 1 en position (i, j) signifie qu'il y a une arête reliant les sommets i et j. Un 0 indique qu'il n'y a aucune arête.

Définition 1.2. Une matrice d'incidences (incidence matrix) Une matrice contenant des 0 et des 1, dont les lignes et les colonnes sont indicéex respectivement par les sommets et les arêtes. Un 1 en position (i, j) signifie que l'arête e_j est incidente au sommet x_i. Un 0 indique le cas contraire.

IV. Sous-graphe, graphe partiel

Définition 1.3. Soit $G = (V, E)$ un graphe, **un sous-graphe** (induced subgraph) de $G = (V, E)$ est un graphe de la forme $G' = (V', E')$ et $V' \subseteq V$ et $E' \subseteq E$ tel que toute arête de E' a ses extrémités dans V'.

Définition 1.4. Un sous-graphe $G' = (V', E')$ de G est dit **couvrant** si $V' = V$. On dit dans ce cas que G' est **un graphe partiel** (spanning subgraph) de G. On peut préciser graphe partiel *engendré* par E', et on note ce sous-graphe $G(E')$.

V. Connexité

Définition 1.5. Soit $G = (V, E)$ un graphe, $u, v \in V$. **Une chaîne** (chain) joignant u et v dans G est une séquence d'arêtes de E telle que :

- La première arête de la séquence est adjacente à u par une de ses extrémités et à la seconde arête de la séquence par son autre extrémité.
- La dernière arête de la séquence est adjacente à v par une de ses extrémités et à l'avant dernière arête de la séquence par son autre extrémité.
- Chaque arête intermédiaire de la séquence est adjacente au précédente par une de ses extrémités et au suivante par l'autre extrémité.

- La **longueur d'une chaîne** est le nombre d'arêtes qu'elle comporte.
- Une chaîne est **simple** si la séquence d'arêtes qui la constituent ne comporte pas plusieurs

fois le même élément. Une chaîne est **élémentaire** si elle n'utilise pas deux fois un même sommet.

Définition 1.6. Un cycle (cycle) est une séquence circulaire d'arêtes **toute distinctes** telle que chaque arête de la séquence soit adjacente à l'arête précédente par une de ses extrémités et à l'arête suivante par l'autre extrémité.
- Un cycle est **élémentaire** si chacun de ses sommets est adjacent à deux arêtes de la séquence au plus.

Proposition 1. *Un cycle est élémentaire si et seulement s'il est minimal (c'est-a-dire si on ne peut en déduire un autre cycle par suppression d'arêtes).*

Proposition 2. *Tout cycle est la somme de cycles élémentaires sans arêtes communes.*

Définition 1.7. Dans un graphe, une **composante connexe** (connected component) est un sous-graphe maximal connexe.
- Maximal signifie qu'il n'y a pas de sous-graphe connexe plus grand contenant les sommets de la composante.

Définition 1.8. Un graphe G est **connexe** (connected) si chaque sommet est accessible à partir de n'importe quel autre. Autrement dit, deux quelconques des sommets sont reliés par une chaîne.
- Un graphe qui n'est pas connexe est dit non connexe, et se décompose en composantes connexes.

Proposition 3. *Si un graphe contient un graphe partiel connexe, il est lui même connexe.*

Définition 1.9. Un isthme (isthmus) d'un graphe G est une arête e dont la suppression augmente le nombre de composantes connexes.

Lemme 1.1. *Une arête d'un graphe G est un **isthme** si et seulement si elle n'appartient pas à un cycle de G.*

Définition 1.10. Un point d'articulation (articulation point) d'un graphe connexe G est un sommet dont la suppression déconnecte le graphe.

1.1.2 Arbres

I. Propriétés des arbres

Théorème 1.2. *l'adjonction d'une arête e à un graphe G a pour effet :*

- *Soit de diminuer le nombre de composantes connexes de une unité ; auquel cas l'arête $e = (u, v)$ n'appartient à aucun cycle de $G' = (V, E \cup \{e\})$.*
- *Soit de laisser inchangé le nombre de composante connexes ; auquel cas l'arête e appartient à un cycle de $G' = (V, E \cup \{e\})$.*

Théorème 1.3. *Soit $G = (V, E)$ avec $|V| = n$*

1. *Si G est connexe alors, $|E| \geq n - 1$.*
2. *Si G est sans cycle alors $|E| \leq n - 1$.*

Théorème 1.4. *Soit $G = (V, E)$ un graphe avec $|V| = n$, $n \geq 2$ sommets. Les propriétés suivantes sont équivalentes et caractérisent un arbre.*

1. *G est connexe et sans cycle,*
2. *G est sans cycle et possède $n - 1$ arêtes,*
3. *G est connexe et admet $n - 1$ arêtes,*
4. *G est sans cycle, et en ajoutant une arête, on crée un et un seul cycle,*
5. *G est connexe, et en supprimant une arête quelconque, il n'est plus connexe,*
6. *Il existe une chaîne et une seule entre deux sommets quelconques de G.*

Exemple 1.1.

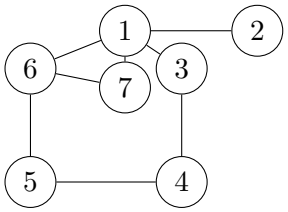


FIGURE 1.1 – Graphe simple et non orienté G

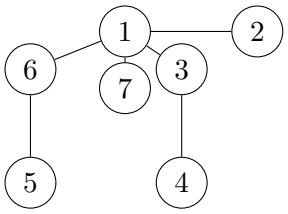


FIGURE 1.2 – Arbre de G

Proposition 1.5. *Un graphe partiel d'un graphe connexe G est un arbre de G si et seulement s'il est connexe et minimal avec cette propriété (relativement à la suppression d'arêtes).*

Proposition 1.6. *Un graphe partiel d'un graphe connexe G est un arbre de G si et seulement s'il est acyclique et maximal avec cette propriété (relativement à l'ajout d'arêtes).*

Les résultats suivants ont un caractère plus technique, mais sont classiques et utiles.

Proposition 4. *Étant donnés un arbre T de G et e une arête de G qui n'appartient pas à T, $T + e$ contient un seule cycle élémentaire.*

Lemme 1.7. *(D'échange) Étant donné un arbre T de G et e une arête de G qui n'appartient pas à T et f une arête du cycle $T + e$, alors $T + e - f$ est un arbre de G.*

Définition 1.11. Les **voisins** d'un arbre sont l'ensemble des arbres qui possèdent $|V| - 2$ arêtes en commun. Étant donnés deux arbres T_1 et T_2 voisins et les deux arêtes e et f les distinguant donc $T_1 \setminus T_2 = \{e\}$ et $T_2 \setminus T_1 = \{f\}$

Définition 1.12. Le complémentaire $E \setminus T$ d'un arbre T est appelé **un co-arbre**, et est noté $\overline{T}$.

D'après le théorème 1.3, pour toute arête $e := uv$ d'un co-arbre T d'un graphe G, il y a une unique chaine dans T connectant ses extrémités. Ainsi $T + e$ contient un unique cycle. Ce cycle est appelé le cycle fondamental de G selon T et e. Par souci de concision, on le note Ce.

Remarque 1.1. On peut tirer d'intéressantes conclusions sur la structure d'un graphe à partir des propriétés de ses cycles fondamentaux selon un arbre.

Théorème 1.8. ***Formule de Cayley***

Le nombre d'arbres dans un graphe complet à n sommets est n^{n-2}.

Définition 1.13. Notation Vectorielle

Les arêtes du graphe étant numérotées de 1 à m, on peut faire correspondre à tout cycle un "vecteur" composé de 1 et 0 de la manière suivante :

- Si l'arête n'appartient pas au cycle, on met un "0",
- Si l'arête appartient au cycle et est parcourue on met un "1".

Définition 1.14. Une **base de cycles** (au sens vectoriel) notée B, est définie comme étant une famille de cycles libres (tous les cycles indépendants, aucun ne peut se définir comme combinaison linéaire des autres) et génératrice (tout cycle peut s'écrire comme combinaison linéaire des cycles de la famille).

Théorème 1.9. *Soit un graphe G d'ordre n (n sommets) avec m arêtes et p composantes connexes, la dimension d'une base des cycles B (appelée «nombre cyclomatique » de G) est donné par :* $|B| = m - n + p$.

Définition 1.15. Une forêt est un graphe sans cycle mais non connexe.

Définition 1.16. Une feuille ou sommet pendant est un sommet de degré 1.

Définition 1.17. Graphe valué (pondéré)

Un graphe valué $G = (V, E)$ est un graphe auquel on adjoint une fonction de valuation. Un graphe peut être valué sur ses sommets comme sur ses arêtes.

Définition 1.18. Soit $G = (V, E)$ un graphe non-orienté. Étant donné $U \subseteq V$, **la coupe** (ou cocycle) $\delta(U)$ (ou $\delta_G(U)$) est définie comme l'ensemble des arêtes de G de la forme $e = uv$ où $u \in V \setminus U$. Lorsque $U = \{u\}$, on écrit simplement $\delta(u)$ à la place de $\delta(\{u\})$. On note $\delta(U) = \delta(U \setminus V)$.

1.2 Problème de l'arbre de poids minimum

Il s'agit d'un problème d'optimisation concernant des objets de base dans les graphes, notamment l'arbre minimum nommé MST (**M**inimum **S**panning **T**ree) qui s'applique à de nombreuses situations concrètes. Nous présentons les algorithmes de Prim [54] et Kruskal [47] qui permettront de trouver en un temps polynomial un arbre de coût minimum dans un graphe non-orienté $G = (V, E)$ (pour des poids quelconques).

1.2.1 Description du problème

Soit $G = (V, E)$ un graphe et soient $\omega_e \in \mathbb{Q}$ $(e \in E)$ des coûts (ou poids) associés à ses arêtes. Pour tout sous-ensemble $T \subseteq E$, le coût de T est défini par :

$$\omega(T) = \sum_{e \in T} \omega_e$$

Nous identifions ici un arbre de G avec un sous-ensemble $T \subseteq E$ pour lequel le graphe partiel (V, T) est un arbre. Le **problème de l'arbre de coût minimum** (**M**inimum **S**panning **T**ree) est de trouver un arbre T de G dont le coût $\omega(T)$ est minimum. Il est clair que le problème a une solution si et seulement si le graphe est connexe ; nous supposons dans la suite qu'il en est ainsi.

Ce problème, qui est l'un des problèmes de base de la recherche opérationnelle, s'applique à un grand nombre de situations concrètes : par exemple, lors de l'élaboration d'un réseau routier ou ferroviaire, ou d'un réseau de lignes électriques ou téléphoniques, etc. [31].

Exemple 1.2.

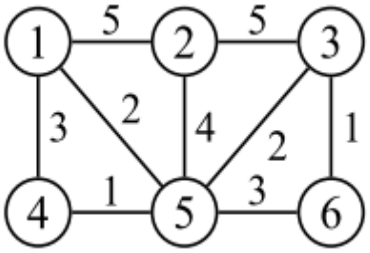


FIGURE 1.3 – Graphe pondéré G

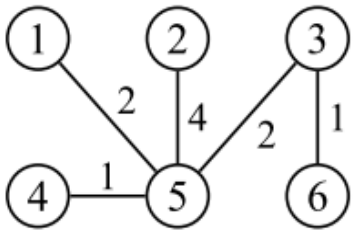


FIGURE 1.4 – Arbre de coût minimum de G

1.2.2 Algorithmes de résolution du problème

Pour la résolution de ce problème, nous proposons les algorithmes de Prim [31, 54], et Kruskal [47, 57].

I. Algorithme de Prim

Il construit itérativement un ensemble de sommets $U \subseteq V$ et un arbre $T \subseteq E$.

début algorithm

- $U := \{v_o\}, T := \emptyset$ ($v_0 \in V$ est un sommet quelconque) ;
- **tant que** $U \neq V$ **faire**
  - prendre $e = \{v_o v_1\}$ une arête de coût minimum dans la coupe $\delta(U)$;
  - $U := U \cup \{v_1\}, T := T \cup \{e\}$;
- **fin tant que**
- retourner T (solution quand $U = V$).

fin algorithme

Temps d'exécution : $O(n^2)$.

II. Algorithme de Kruskal

Après avoir ordonné les arêtes par coûts croissants, il construit progressivement le graphe partiel (V, T) qui est un arbre de coût minimum.

début algorithme

- ordonner les arêtes : $e_1, ..., e_m$ de telle sorte que $\omega_{e_1} \leq ... \leq \omega_{e_m}$;
- $T := \emptyset$;
- **pour** $j := 1$ à m **faire**
  - **si** $(V, T \cup \{e_j\})$ ne contient pas de cycle **alors** $T := T \cup \{e_j\}$

- **fin si**

– **fin pour**

fin algorithme

Temps d'exécution : $O(m\ log\ n)$.

Remarque 1.2. On peut arrêter l'exécution dès que $|T| = n - 1$.

Théorème 1.10. *La solution retournée par chacun des algorithmes de Prim et de Kruskal est un arbre de coût minimum.*

Remarque 1.3. On note que le problème de l'arbre de coût minimum est équivalent au problème de l'arbre de coût maximum si on remplace les coûts ω_e des arêtes $e \in E$ par $-\omega_e$ et les algorithmes précédents sont applicables à ce problème.

1.2.3 Applications du problème

I. Application en réseaux [11]

Ce problème peut se poser lors de l'établissement d'un réseau, qu'il soit de communication ou d'interconnexion, si, ayant estimé le coût des liaisons directes entre toute paire d'objets à relier, on cherche à réaliser un réseau connexe de coût minimum. Il est clair qu'une solution optimale à ce problème est un graphe sans cycle puisque tous les coûts étant positifs, si une solution comportait un cycle, on obtiendrait une solution plus économique en supprimant une arête de ce cycle. Or, on prouve aisément que la suppression d'une arête ne peut déconnecter un graphe connexe. (autrement dit, une arête d'un cycle n'est jamais un isthme).

II. Application en traitement d'image [11]

Une image (en noir et blanc) est présentée sous forme de pixels, définis par les 512 lignes et 512 colonnes d'un réseau. Chaque pixel possède un certain niveau de gris et, en dehors des points situés sur les bords, admet huit voisins. On cherche à déterminer des régions dans cette image, c'est à dire des parties connexes de l'image constituées de points de niveaux de gris très voisins. On modélise la donnée par un graphe, dont presque tous les sommets sont de degré 8 : chaque sommet du graphe correspond à un des 262144 pixels

de l'image, deux sommets sont adjacents si et seulement si les pixels qu'ils représentent sont voisins, et on value l'arête qui les joint par un nombre proportionnel à la différence de leurs niveaux de gris.

Moriss et Constantinidès ont proposé une heuristique, c'est à dire une méthode approchée, pour déterminer les région d'une image : construire un arbre de coût minimum puis supprimer de cet arbre les arêtes de coût supérieur à un seuil donné, fixé de façon à séparer deux pixels dont les niveaux de gris sont trop différents pour qu'ils appartiennent à la même zone. On obtient ainsi une forêt couvrante, dont les composantes connexes qui sont des arbres, couvrent ce que l'on considèrera comme les zones cherchées.

Chapitre 2

Programmation linéaire

Ce chapitre a pour objectif principal de présenter le contexte de l'optimisation linéaire. Nous rappelons en premier lieu des éléments essentiels de la théorie de la programmation linéaire en variables aussi bien continues qu'entières. Nous introduisons ensuite, les notions fondamentales ainsi que les principaux résultats liés à la théorie de la programmation mon-objectif linéaire en nombres entiers.

2.1 Résultats fondamentaux de la programmation linéaire

La programmation linéaire est la technique la plus célèbre et l'outil le plus puissant de la recherche opérationnelle, depuis longtemps elle a prouvé sa valeur comme un modèle important pour de nombreux problèmes d'allocation et des phénomènes économiques. Les applications en expansion continue dans la littérature, ont démontré à maintes reprises l'importance de la programmation linéaire comme un cadre général pour la formulation de problèmes. En effet, un programme mathématique est un problème d'optimisation d'une fonction objectif de plusieurs variables en présence de contraintes. Le programme est dit linéaire si la fonction et les contraintes sont toutes des combinaisons linéaire de variables. Ce chapitre est principalement basé sur les livres : [36, 56, 57, 61]

Modélisation d'un programme linéaire

La formalisation d'un programme est une tâche délicate mais essentielle car elle conditionne la découverte ultérieure de la bonne solution. Elle comporte les mêmes phases quelles que soient les techniques requises ultérieurement pour le traitement (programmation linéaire

ou programmation non linéaire) :

1. La détection du problème et l'identification des variables. Ces variables doivent correspondre exactement aux préoccupations du responsable de la décision. En programmation mathématique, les variables sont des variables décisionnelles.

2. La formulation de la fonction objectif traduisant les préférences du décideur exprimées sous la forme d'une fonction des variables identifiées.

Définition 2.1. On appelle problème d'optimisation linéaire sous forme générale un problème de la forme :

$$\begin{cases} max\ Z(x) \\ G_1(x) \leq 0 \\ G_2(x) \leq 0 \\ \vdots \\ G_m(x) \leq 0 \\ x \in \mathbb{R}^n. \end{cases} \tag{2.1.1}$$

Où $n, m \in \mathbb{N}^*, Z : \mathbb{R}^n \to \mathbb{R}$ est une fonction linéaire et $G_i, \forall i = 1, ..., m$ sont des applications affines définies sur $\mathbb{R}^n$ et à valeurs réelles .

Définition 2.2. On dit que $\overline{x} \in \mathbb{R}^n$ est une solution réalisable du problème (2.1.1) si $\overline{x}$ vérifie le système d'inéquations précédent, autrement dit si $G_i(\overline{x}) \leq 0, \forall i = 1, ..., m$. L'ensemble S de toutes les solutions réalisables d'un problème d'optimisation est appelé son ensemble réalisable.

Définition 2.3. On dit que $x^* \in \mathbb{R}^n$ est une solution optimale du problème (2.1.1) si x^* est une solution réalisable du problème (2.1.1) et si de plus, quelle que soit la solution réalisable $y \in \mathbb{R}^n$ du problème (2.1.1), on a nécessairement $Z(y) \leq Z(x^*)$. Autrement dit, une solution réalisable est optimale si elle maximise la fonction objectif sur l'ensemble des solutions réalisables.

Remarque 2.1. L'ensemble S est un polyèdre dans $\mathbb{R}^n$, c'est-à-dire une intersection finie de demi-espaces fermés de $\mathbb{R}^n$. Si S est de plus non vide et borné (cela n'est pas nécessairement le cas), et puisque la fonction objectif est une fonction continue, l'existence d'au

moins une solution optimale est alors garantie. Dans tous les cas, nous verrons que si la fonction est majorée sur S, alors le problème de maximisation a toujours au moins une solution optimale.

2.1.1 Formes matricielles classiques

Notons $x = (x_1, x_2, ..., x_n)^t$ le vecteur des variables, $b = (b_1, b_2, ..., b_m)^t$ celui des seconds membres des contraintes, $c = (c_1, c_2, ..., c_n)$ les coûts ou profils associés aux variables, et $A = a_(ij)$ la $m \times n$-matrice des contraintes. Deux formes matricielles sont courantes : la forme canonique avec des contraintes $Ax \leq b$ et la forme standard avec des contraintes d'égalités, pour la résolution algébrique par des programmes. Par convention, la forme standard est exprimée avec les seconds membres positifs.

Un problème sous forme canonique s'écrit donc :

$$\begin{cases} max\ Z = \quad c^t x \\ Ax \leq b \\ x \geq 0 \end{cases} \tag{2.1.2}$$

En écriture matricielle, un problème sous forme standard s'écrit donc

$$\begin{cases} max\ Z = \quad c^t x \\ Ax = b \\ x \geq 0 \end{cases} \tag{2.1.3}$$

Remarque 2.2. - Dans la réalité, un PL peut comporter à la fois des égalités et des inégalités, on peut donc convertir les formes mixtes en formes classiques. Ainsi, toute contrainte d'égalité peut être remplacée par des inégalités ($\leq$ et $\geq$).

- On peut convertir une inégalité en égalité en ajoutant une variables d'écart, propre à chaque contrainte.

- On peut passer d'une maximisation à une minimisation, car maximiser Z revient à minimiser $-Z$.

Théorème 2.1. *Tout programme linéaire peut être écrit sous forme canonique ou sous*

forme standard.

Remarque 2.3. Sans perte de généralité, on peut supposer que dans un problème sous forme standard, les lignes de A sont linéairement indépendantes (si ce n'est pas le cas soit certaines contraintes sont redondantes, soit l'ensemble des contraintes est vide). Dans la suite, lorsque nous parlerons de problème sous forme standard, nous supposerons implicitement que les lignes de A sont linéairement indépendantes, autrement dit que $rang(A) = m$.

Définition 2.4. Considérons le problème d'optimisation linéaire sous forme standard

$$(P)\begin{cases} max\ Z = \quad c^t x \\ Ax = b \\ x \geq 0 \end{cases} \tag{2.1.4}$$

Définition 2.5. Soit $B \subset \{1, 2, ..., n\}$ un ensemble d'indices avec $card(B) = m$ tel que les colonnes $A^j, j \in B$, de A sont linéairement indépendantes. On dit que l'ensemble des colonnes $A^j, j \in B$, est une base et les indices de B sont appelés indices de base.

- les variables $x_B = (x_j, j \in B)$ sont appelées variables de base.
- les variables $x_N = (x_j, j \notin B)$ sont appelées variables hors base.

Définition 2.6. On dit que $x = (x_B, x_N)$ est solution de base associée à la base B si $x_N = 0$. $x = (x_B, x_N)$ est une solution de base réalisable si $x_N = 0$ et $A_B x_B = b$, $x_B = A_B^{-1} b \geq 0$. Dans ce cas, on dit que la base B est réalisable.

Définition 2.7. Étant donnée une base réalisable B du programme linéaire (P), le programme linéaire est équivalent à (P') :

$$\begin{cases} \hat{c}^N x_N = z(max) - \pi b \\ x_B + A_B^{-1} A^N x_N = A_B^{-1} b \quad x \geq 0 \end{cases}$$

Le problème (P') est dit forme canonique de (P) par rapport à la base B où :

- $N = \{1, 2, ..., n\} \setminus B$
- $\pi = c^B A_B^{-1}$ est dit vecteur multiplicateur relatif à la base B
- $\hat{c} = c - \pi A$ est dit vecteur des coûts réduits relatif à la base B

Théorème 2.2. *Si le vecteur des coûts réduits ĉ relatif à une base réalisable B est négatif ou nul, la solution de base correspondante est une solution optimale de (P). La base B est alors dite base optimale.*

2.1.2 Méthode de résolution d'un programme linéaire

Un problème linéaire peut être résolu en temps polynomial. L'algorithme du simplexe [17] est le plus célèbre (et le plus efficace dans le cas général) des algorithmes de résolution, bien qu'il ne soit pas polynomial !. L'algorithme du simplexe repose sur le fait qu'une solution optimale d'un programme linéaire peut être prise parmi les sommets du polyèdre de $\mathbb{R}^n$ déterminé par $Ax \leq b$. On donne les étapes de l'algorithme du simplexe et on détaille la (PLNE) dans la section suivante.

Algorithme du simplexe

L'algorithme du simplexe, introduit en 1947 par G.B. Dantzig [17], décrit un moyen intelligent de se déplacer d'une solution de base admissible à une autre améliorant la valeur de la fonction objectif, jusqu'à trouver une solution optimale en un nombre fini d'étapes. Malgré une complexité théorique dans le pire des cas exponentielle, il permet de résoudre la plupart des problèmes non générés aléatoirement rapidement.
Soit à résoudre le programme linéaire :

$$(P)\begin{cases} Max \quad z = c^t x \\ Ax = b \\ x \geq 0 \end{cases}$$

On définit l'application linéaire *col* par :

$$\begin{array}{rccl} col: & \{1,\ldots,m\} & \longrightarrow & \{1,\ldots,n\} \\ & i & \longrightarrow & col(i) = \text{indice de la variable de base associée à la ligne } i. \end{array}$$

1. ***Écriture canonique.*** Soit B une base réalisable de départ, alors l'écriture canonique de (P) par rapport à B donne :

1	0	$\cdots$	0	$\widehat{A}^N = (A^B)^{-1}A^N$	$\widehat{b} = (A^B)^{-1}b$
0	$\ddots$		$\vdots$		
$\vdots$		$\ddots$	0		
0	$\cdots$	0	1		
0	$\cdots$	0	0	$\widehat{c}_N = c_N - c_B(A^B)^{-1}A^N$	$z - c_B(A^B)^{-1}b$

TABLE 2.1 – Écriture canonique de $(P)/B$

2. ***Choix de la colonne pivot.*** (variable à entrer en base)
   (a) Si $\forall j \in N$, $\widehat{c}_j \leq 0$, alors **STOP** (la solution optimale est trouvée).
   (b) Sinon, choisir une colonne s telle que : $\widehat{c}_s = \underset{j \in N}{max}\, \widehat{c}_j$
3. ***Choix de la ligne pivot.*** (variable à sortir de la base)
   (a) Si $\forall i = \overline{1,m}$, $\widehat{A}_i^s < 0$, alors **STOP** (la fonction objectif n'est pas bornée).
   (b) Sinon, choisir une ligne r, telle que :

$$\frac{\widehat{b}_r}{\widehat{A}_r^s} = \underset{i=\overline{1,m}}{min} \left\{ \frac{\widehat{b}_i}{\widehat{A}_i^s} | \widehat{A}_i^s > 0 \right\}$$

$$B = B \cup \{s\} \setminus col(r)$$

4. ***Opération pivot.*** (passage au tableau suivant) : Soient $L_1, L_2, \ldots, L_m$ les m premières lignes du tableau 2.1 correspondants aux contraintes du problème et L_{m+1} la $(m+1)$ ème ligne correspondant à la fonction objectif, alors les lignes du nouveau tableau sont calculées ainsi :
   (a) $L_i \leftarrow L_i - \dfrac{L_r \widehat{A}_i^s}{\widehat{A}_r^s} \quad i = \overline{1,m},\ i \neq r$
   (b) $L_{m+1} \leftarrow L_{m+1} - \dfrac{L_r \widehat{c}_s}{\widehat{A}_r^s}$
   (c) $L_r \leftarrow \dfrac{L_r}{\widehat{A}_r^s}$.

Retour à (1)

On distingue dans la programmation linéaire (PL), la programmation en nombres entiers (PLNE), pour laquelle les variables sont astreintes à être entières, si les variables ne peuvent prendre que les valeurs 0 ou 1, le programme est linéaire à variables bivalentes,

binaires ou de décision, et la programmation linéaire en nombres réels, pour laquelle les variables sont continues. Bien entendu, il est possible d'avoir les deux en même temps (variables entières et continues), on parle alors, de programme linéaire mixte. Cependant, à partir du moment où au moins une contrainte ou la fonction objectif n'est plus une combinaison linéaire de variables, on a affaire à un programme non linéaire (PNL). Les $PLNE$ et PL en 0-1 sont plus difficiles à résoudre que les PL ordinaires, et les PNL sont encore plus difficiles et les algorithmes actuels ne trouvent en général qu'un optimum local. La difficulté d'un problème découle donc de l'efficacité des algorithmes de résolution qu'on peut lui appliquer.

2.2 Programmation linéaire en nombres entiers (PLNE)

La programmation linéaire en variables entières est très importante, car la plupart des problèmes réels comportent des variables qui doivent, par nature, prendre nécessairement une valeur entière. La programmation linéaire en nombres entiers étudie des programmes linéaires dans lesquels les variables sont astreintes à être entières. En particulier, les variables peuvent être simplement binaires. De nombreuses contraintes, en apparence non linéaires, peuvent être linéarisées grâce à des variables entières. Ces possibilités augmentent énormément le champ d'application de la programmation linéaire. Même si les programmes linéaires obtenus sont souvent difficiles à résoudre, la programmation linéaire en nombres entiers est déjà très utile comme langage de modélisation : elle permet de décrire de façon concise des problèmes d'optimisation discrète. [36, 56, 58, 67]

Un problème de la programmation linéaire multiobjectif en nombres entiers peut être formulé comme suit :

$$\begin{cases} max\ z(x) \\ x \in S,\ x \text{ vecteur entier} \end{cases} \tag{2.2.1}$$

où x est un vecteur de n variables entières positives ou nulles, z est une fonction scalaire et S l'ensemble des contraintes défini comme suit : $S = \{x \in \mathbb{R}^n / Ax \leq b, x \geq 0\}$, tel que $A \in Z^{m,n}, b \in Z^m$.

Le $PLNE$ est un problème NP-difficiles, car de nombreux problèmes NP-difficiles peuvent être exprimés comme des $PLNE$ (par exemple trouver un stable dans un graphe

$G = (V, E)$ revient à trouver un vecteur $x \in \{0,1\}^E$ satisfaisant $x_u + x_v \leq 1$ pour toute arête $uv \in E$).

Remarque

Les remarques suivantes soulignent les difficultés de $PLNE$:

- L'optimum entier n'est pas forcément un sommet du polyèdre, il doit être à l'intérieur du polyèdre.

- Si le PL relaxé avait un optimum entier, ce serait aussi l'optimum du $PLNE$.

- Sinon, le coût optimal du PL relaxé donne une borne supérieure (une bonne inférieure en minimisation) du coût optimal du $PLNE$.

- L'arrondi de la solution du PL relaxé n'est pas nécessairement optimal pour le $PLNE$, il pourrait même ne pas être réalisable.

- Le polyèdre peut être non vide mais, n'admettre aucune solution entière.

2.2.1 Quelques méthodes de résolution des PLNE

Les deux principales familles de méthodes actuellement connues pour résoudre les problèmes de programmation linéaire en nombres entiers sont les méthode arborescentes et les méthodes des coupes. On présente ci-dessous la méthode : "Branch and Bound" [48] et la méthode des Coupes de Gomory [38].

2.2.2 "Branch and Bound"

La méthode par séparation et évaluation ("Branch & Bound) est très efficace pour la résolution des problèmes de programmation linéaire en nombres entiers. Elle a été à l'origine développée par Land et Doig (1960) [48] et a été modifiée plus tard par Dakin (1965) [19].

L'algorithme consiste à séparer de manière récursive le problème en sous-problèmes de cardinalité inférieure tant que la résolution de ces problèmes reste difficile. Le cardinal de l'ensemble à explorer est réduit en imposant à cet ensemble des contraintes supplémentaires (réduction du domaine). Une série de tests, appliquée à tous les sous-problèmes permet de supprimer de l'espace de recherche les sous-problèmes qui ne peuvent pas engendrer de solution optimale.

Cette recherche par décomposition de l'ensemble des solutions peut être représentée

graphiquement par un arbre. C'est de cette représentation que vient le nom de méthode de recherche arborescente .

- Chaque sous-problème créé au cours de l'exploration est symbolisé par un nœud de l'arbre (ou sommet), le nœud racine représentant le problème initial.

- Les branches de l'arbre symbolisent le processus de séparation, ils représentent la relation entre les nœuds.

- Les nœuds non séparés sont appelés nœuds pendants (par exemple, $(S1)$, $(S3)$ et $(S4)$, de la figure 2.1 :

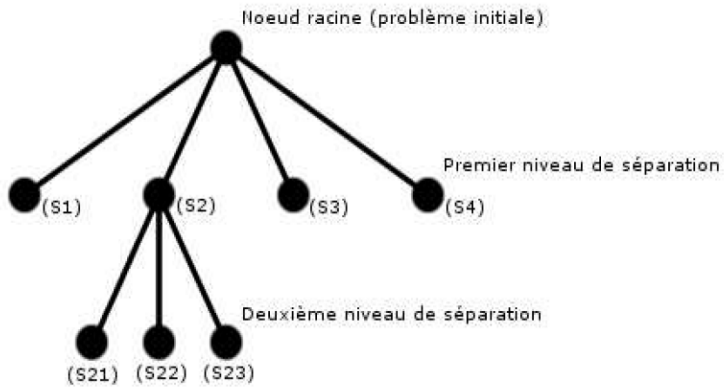


FIGURE 2.1 – Exemple d'arbre de recherche

Un arbre construit durant la résolution afin de représenter toutes les combinaisons de valeurs possibles pour les variables binaires est présenté dans la figure 2.2 :

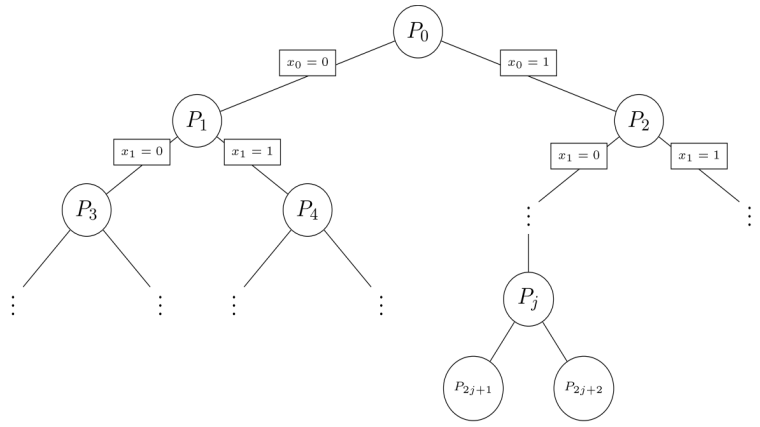


FIGURE 2.2 – Exemple d'arbre binaire

Chaque nœud de l'arbre ci-dessus représente un sous-problème pour lequel certaines va-

riables binaires sont fixées. Les variables dont la valeur n'est pas encore fixée sont appelées variables libres. Le nœud racine représente le problème originel à résoudre sans variable binaire fixée et les feuilles de l'arbre correspondent aux sous-problèmes pour lesquels toutes les variables binaires sont fixées.
Le "Branch and Bound" est basé sur trois axes principaux :

- **L'évaluation**

  L'évaluation permet de réduire l'espace de recherche en éliminant quelques sous ensembles qui ne contiennent pas la solution optimale. L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence. Le "Branch and Bound" utilise une élimination de branches dans l'arborescence de recherche de la manière suivante : la recherche d'une solution de coût minimal, consiste à mémoriser la solution de plus bas coût rencontré pendant l'exploration, et à comparer le coût de chaque nœud parcouru à celui de la meilleure solution. Si le coût du nœud considéré est supérieur au meilleur coût, on arrête l'exploration de la branche et toutes les solutions de cette branche seront nécessairement de coût plus haut que la meilleure solution déjà trouvée. La partie évaluation consiste à calculer une borne inférieure lb et une borne supérieure ub pour un ensemble donné, de sorte que la borne $lb \leq ub$. Une borne inférieure peut être la valeur de la meilleure solution trouvée jusqu'à cet instant de la recherche, ou la valeur d'une solution obtenue avec une méthode approchée. La borne supérieure est utilisée pour fermer certaines régions de l'espace à explorer. Une bonne borne supérieure doit être rapide à calculer tout en étant serrée. Cependant, ces deux qualités s'opposent en pratique.

- **La séparation**

  La séparation consiste à diviser le problème en deux sous-problèmes selon les valeurs d'une variable déjà choisie x. Pour un $PLNE$ général, on sépare en considérant un entier p et les deux sous-problèmes $x \leq p$ et $x \geq p+1$. Pour un PL en 0-1, on sépare en considérant les deux cas $x = 0$ et $x = 1$. Les $PLNE$ des sous-problèmes peuvent à leur tour être séparés, ce qui forme progressivement une arborescence dont chaque nœud correspond à un sous-problème. Cette arborescence peut être énorme. Ainsi,

en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer toutes les solutions. L'ensemble de nœuds de l'arbre qu'il reste encore à parcourir comme étant susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser, est appelé ensemble des nœuds actifs.

– **Les stratégies de parcours**

La largeur d'abord : Cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les deux autres stratégies présentées.

La profondeur d'abord : Cette stratégie avantage les sommets les plus éloignés de la racine (de profondeur la plus élevée) en appliquant plus de séparations au problème initial. Cette voie mène rapidement à une solution réalisable en économisant la mémoire.

Le meilleur d'abord : Cette stratégie consiste à explorer des sous-problèmes possédant la meilleure borne. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une mauvaise évaluation par rapport à la valeur optimale.

2.2.3 Coupes de Gomory

L'algorithme utilisant les coupes de Gomory procède comme suit :

Soit $\overline{x}$ une solution optimale du problème relaxé (PL) trouvée par la méthode du simplexe. Si $\overline{x}$ est une solution entière, elle est optimale pour $(PLNE)$. Sinon, choisir une variable x_j telle que la valeur $\overline{x}_j$ est fractionnaire et considérer la ligne correspondante du tableau du simplexe, par exemple la ligne i :

$$x_j + \sum_{k \in N} \widehat{a}_{ik} x_k = \overline{x}_j$$

où N est l'ensemble des indices des variables hors-base.

La contrainte

$$\sum_{k \in N} f(\widehat{a}_{ik}) x_k \geq f(\overline{x}_j) \tag{2.2.2}$$

est alors déduite de l'expression précédente, où $f(r) = r - \lfloor r \rfloor$ désigne la partie fractionnaire du nombre réel r.

Cette coupe, appelée coupe fractionnaire de Gomory, ou coupe fondamentale, peut être rajouter au tableau courant du simplexe. Dans la pratique, les coupes fractionnaires de Gomory convergent très lentement. D'autres coupes ont été introduites dans l'intention de les rendres plus performantes. En particulier, Gomory [38] lui même a donné les coupes suivantes :

Proposition 5. *L'inéquation*

$$\sum_{k \in N} min\{f(\widehat{a}_{ik}), f(\overline{x}_j)\frac{1 - f(\widehat{a}_{ik})}{1 - f(\overline{x}_j)}\}x_k \geq f(\overline{x}_j) \tag{2.2.3}$$

est une coupe. De plus, elle est plus profonde que la coupe 2.2.2

De même, pour tout entier naturel t, l'inéquation

$$\sum_{k \in N} min\{f(t\widehat{a}_{ik}), f(t\overline{x}_j)\frac{1 - f(t\widehat{a}_{ik})}{1 - f(t\overline{x}_j)}\}x_k \geq f(t\overline{x}_j) \tag{2.2.4}$$

est une coupe plus profonde que (**??**)

Remarque 2.4. Cette méthode a un sérieux inconvénient qui est associé aux erreurs d'arrondis qui surgissent pendant les calculs numériques. En raison de ces erreurs d'arrondis, nous pouvons finalement obtenir de faux résultats concernant le test d'intégrité sur les variables. D'autre part, le nombre de coupes de Gomory [38] à générer peut être très grand. Ce qui fait augmenter la taille du problème sans limite, puisqu'une variable d'écart et une contrainte sont ajoutés à chaque fois que $\widehat{b}$ n'est pas entier où $\widehat{b}$ est le vecteur des seconds membres des contraintes du problème après les opérations de pivotage.

Chapitre 3

Programmation linéaire multiobjectif

De nombreux problèmes rencontrés quotidiennement dans divers domaines (transport, télécommunication, économie ...) peuvent être modélisés sous la forme de programmes linéaires en nombres entiers et sont étudiés dans le cadre de l'optimisation combinatoire. De nombreuses méthodologies, exactes ou approchées, ont été proposées et il est aujourd'hui possible de résoudre des problèmes de taille conséquente.

Par ailleurs, la prise en compte simultanée de plusieurs objectifs contradictoires est de plus en plus considérée et forme l'objet d'étude de l'optimisation combinatoire multiobjectif. Les méthodes pour l'optimisation multiobjectif ont connu un intérêt croissant ces vingt dernières années. Cependant, le domaine de recherche a tendance à s'enliser, privilégiant des méthodes basées sur les métaheuristiques tels que les algorithmes évolutionnaires. Alors que l'on recense des centaines d'études dédiées aux algorithmes évolutionnaires multiobjectif, le nombre portant sur les méthodes exactes multiobjectif est de l'ordre de la dizaine.

Ainsi, un effort reste à fournir pour amener les méthodes multiobjectif au même niveau d'utilisabilité et de performance que les méthodes en optimisation classique, et ainsi développer l'adoption et l'utilisation de l'optimisation combinatoire multiobjectif, en leur permettant de s'attaquer à des problèmes réalistes de grande taille.

Le projet de cette thèse sera d'explorer l'adaptation et l'élaboration de techniques,

notamment des méthodes exactes comme les algorithmes de séparations en utilisant la programmation mathématique pour l'optimisation multiobjectif, en se focalisant sur les problèmes linéaires en nombres entiers et plus particulièrement les problèmes en variables 0-1.

Les problèmes d'optimisation combinatoire issus des problématiques réelles sont la plupart du temps de nature multiobjectif, car plusieurs critères d'évaluation souvent contradictoires sont à considérer simultanément. Optimiser un tel problème relève donc de l'optimisation combinatoire multiobjectif.

L'optimisation multiobjectif possède ses racines dans les travaux en économie de Edgeworth [24] et Pareto [52]. Elle a ainsi été initialement utilisée en économie et dans les sciences du management, puis graduellement dans les sciences de l'ingénieur.

Pourtant, malgré l'intérêt indéniable de la modélisation et de la résolution multiobjectif des problèmes rencontrés dans l'industrie, dans les télécommunications et d'autres domaines de la vie quotidienne, peu de travaux ont été réalisés en optimisation combinatoire multiobjectif avant les années 80-90. Mais depuis,un fort intérêt a été montré pour l'aide à la décision multiobjectif qui consiste pour un problème comportant plusieurs objectifs, à déterminer, parmi les solutions de meilleur compromis entre les objectifs, la solution la plus intéressante pour le problème en question. Ainsi, une phase importante concerne l'optimisation multiobjectif qui recherche les solutions de compromis.

L'objet de ce chapitre est de présenter quelques notions fondamentales concernant les problèmes de programmation linéaire en nombres entiers dans le cas où plusieurs objectifs à optimiser sont à considérer.

3.1 Sur la programmation multiobjectif linéaire discrète

Un problème d'optimisation mono-objectif peut être formulé comme suit :

$$(PL)\begin{cases} maximiser & z(x) \\ sous & x \in S \end{cases} \tag{3.1.1}$$

où x est un vecteur de n variables, z est une fonction scalaire et S l'ensemble des

solutions réalisables défini par des contraintes comme suit : $S = \{x \in \mathbb{R}^n / Ax \leq b, x \geq 0\}$, A est une $m \times n$-matrice réelle et b un m-vecteur réel.

La formulation précédente était relative à un problème dans lequel on recherchait un optimum pour une fonction objectif. Cependant, lorsque l'on modélise un problème, on cherche souvent à satisfaire $k, (k > 1$ et entier) objectifs. Par exemple, on veut un système performant et on veut aussi que ce système consomme peu. Dans ce cas, on parle de problème d'optimisation multiobjectif (ou problème d'optimisation multicritère). Celui-ci s'écrit de la manière suivante :

$$(P) \begin{cases} maximiser & z(x) = (z_1(x), z_2(x), ..., z_k(x)) \\ sous & x \in S \end{cases} \tag{3.1.2}$$

où $S = \{x \in \mathbb{R}^n / Ax \leq b, x \geq 0\}$ est l'ensemble des solutions réalisables de (P) (appelée aussi **espace de décision**) définies ci-dessus. A chaque solution réalisable x dans S, on associe son image $z(x) = (z_1(x), z_2(x), ..., z_k(x))$ dans $\mathbb{R}^k$ (**espace des critères**) et on construit donc, l'ensemble $C = \{y \in \mathbb{R}^n : y = z(x), x \in S\} = z(S)$.
Sans perte de généralité nous supposerons par la suite que nous considérons des problèmes de maximisation.

3.1.1 Terminologie - Définitions

I. Relation de dominance et solutions efficaces [66, 70]

Pour qu'une solution $x \in S$ soit un possible compromis satisfaisant, il apparaît naturel d'imposer qu'il n'existe aucune autre solution admissible $y \in S$ qui fournisse des valeurs au moins aussi bonnes que celles de x sur chaque critère et même meilleur sur au moins un critère. C'est une condition minimale à satisfaire qui justifie les définitions suivantes :

Définition 3.1. - Un point $z \in \mathbb{R}^n$ domine un point $z' \in \mathbb{R}^n$, si et seulement si $\forall i = 1...k$, $z_i \geq z'_i$ et $\exists i \in \{1...k\}$ tel que $z_i > z'_i$, (c-à-d $z \neq z'$).
- Un point $z \in \mathbb{R}^n$ est non-dominé s'il n'existe aucun point $z' \in \mathbb{R}^n$ qui le domine.
- Nous noterons $Z(S)$ l'ensemble de ces vecteurs.

Définition 3.2. Une solution $x \in S$ est efficace (ou Pareto optimale) si son image $z(x)$

est non-dominée. L'ensemble des solutions efficaces est noté $E(P)$.

Définition 3.3. - Le front Pareto est, dans l'espace des critères, l'ensemble correspondant des points non dominés.
- L'ensemble des solutions efficaces est l'ensemble des solutions intéressantes du point de vue multicritère. Il est cependant généralement de grande cardinalité et même, pour un problème en variables continues, infini non dénombrable.

Résoudre le problème (P) revient à trouver, soit l'ensemble des solutions efficaces dans l'espace des décisions, soit l'ensemble des solutions non dominées dans l'espace des critères.

II. Point idéal, matrice des gains et point "nadir"

Définition 3.4. Le point idéal $z^* = (z_1^*, z_2^*, ..., z_n^*)$ est le vecteur qui optimise chacune des fonctions objectifs $z_i = c_i x$, i.e : $z_i^* = min\{c_i x \backslash x \in S\}$. Les coordonnées de ce point sont obtenues en optimisant chaque fonction objectif séparément.

Définition 3.5. La matrice des gains associée au problème (P) peut être représentée par une matrice carrée G de dimension $k(k > 1)$ comme suit :

$$\begin{pmatrix} z_1^* & z_{12} & \dots & z_{1k} \\ z_{21} & z_2^* & \dots & z_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ z_{31} & z_{32} & \dots & z_k^* \end{pmatrix} \tag{3.1.3}$$

avec $z_i^* = \min\limits_{x \in D} z_i = c_i x_i^*$, $\forall i = 1, ..., k$, et $z_{ij} = c_i x_j^*$, $\forall i = 1, ..., k$, $\forall j = 1, ..., k$, avec $i \neq j$.

Il faut noter que cette matrice dépend de la solution optimale x_i^* et peut donc, ne pas être unique.

Définition 3.6. Le point "nadir" $N \in \mathbb{R}^k$: $n_i = \max\{c_i x \backslash x$ efficace pour $(P)\}$. Les coordonnées de ce point correspondent aux pires valeurs obtenues par chaque fonction objectif lorsque l'on restreint l'espace des solutions à la surface de compromis.

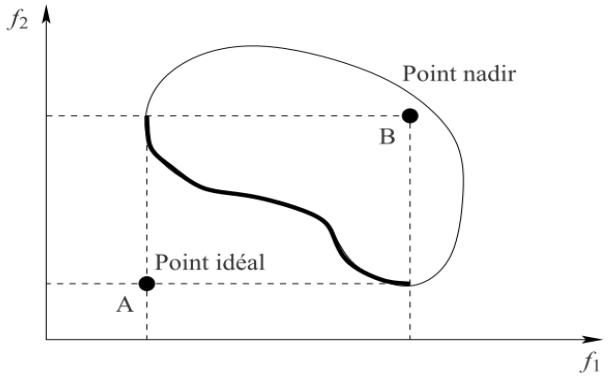


FIGURE 3.1 – Représentation du point idéal et du point "nadir".

III. Convexité

Définition 3.7. Un ensemble S est convexe si, étant donnés deux points distincts quelconques de cet ensemble, le segment qui relie ces deux points est contenu dans l'ensemble S.

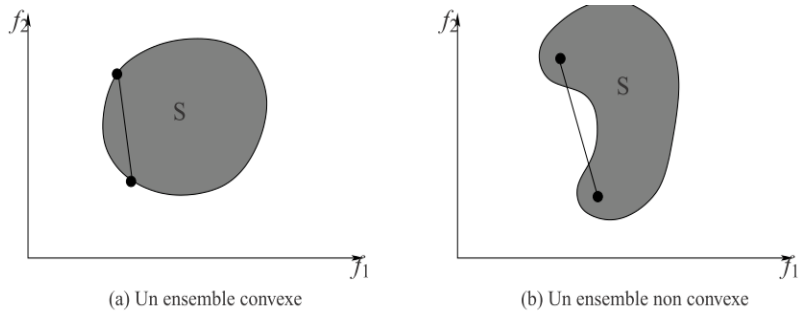


FIGURE 3.2 – Exemples d'ensemble convexe et d'ensemble non convexe.

3.1.2 Notion de cônes

Définition 3.8. Soit $u \in U \subset \mathbb{R}^n$, $U \neq \emptyset$. Alors, U est un *cône* si et seulement si $\alpha u \in U$ pour tout scalaire $\alpha \geq 0$. L'origine $0 \in \mathbb{R}^n$ est contenu dans chaque cône.

Excepté de l'ensemble singleton qui contient uniquement l'origine, tous les cônes sont non bornés. Comme exemple, le demi espace fermé $\left\{x \in \mathbb{R}^n | c^t x \leq 0\right\}$ est un cône mais le demi espace ouvert $\left\{x \in \mathbb{R}^n | c^t x > 0\right\}$ n'est pas un cône car il ne contient pas l'origine. [66]

Définition 3.9. Considérons $\left\{u^1, u^2, \ldots, u^k\right\}$, un ensemble de k vecteurs de $\mathbb{R}^n$ et l'en-

semble U tel que :

$$U = \left\{ u \in \mathbb{R}^n | u = \sum_{i=1}^{k} \alpha_i u^i,\ \alpha_i \geq 0 \right\},$$

U est l'ensemble de toutes les combinaisons linéaires à coefficients non négatifs des u^i, $i = \overline{1,k}$, et il est le cône convexe engendré par l'ensemble $\{u^1, u^2, \ldots, u^k\}$. Les vecteurs u^i, $i = \overline{1,k}$ sont appelés les *générateurs* de U.

Le seul cône pour lequel l'ensemble des générateurs est unique est $\{0 \in \mathbb{R}^n\}$.
Soit $\{u^1, u^2, \ldots, u^k\}$ un ensemble de générateurs pour le cône convexe U et soit $u^l \in \{u^1, u^2, \ldots, u^k\}$. Alors, u^l est *générateur non essentiel* si U peut être généré par $\{u^1, u^2, \ldots, u^k\} \setminus \{u^l\}$. Un générateur non essentiel est celui qui peut être exprimé comme combinaison linéaire d'autres générateurs, il est dit *essentiel* sinon.

Définition 3.10 (**Dimension d'un cône**)**.** La *dimension* d'un cône $U \subset \mathbb{R}^n$ est donné par le nombre de vecteurs linéairement indépendants de U.

Par exemple, la dimension du cône singleton $\{0 \in \mathbb{R}^n\}$ est 0, et la dimension d'un cône convexe généré par un ensemble de k vecteurs linéairement indépendants est k. On peut déterminer la dimension d'un cône en calculant le rang de la matrice dont les lignes (ou les colonnes) sont les générateurs de ce cône.

Définition 3.11. Soit $U \subset \mathbb{R}^n$ un cône. Alors, le *cône polaire* non négatif de U (noté $U^{\geq}$) est le cône convexe :

$$U^{\geq} = \{ y \in \mathbb{R}^n | y^t u \geq 0 \text{ pour tout } u \in U \},$$

c'est à dire, tous les vecteurs de $U^{\geq}$ font un angle inférieur ou égal à 90^0 avec chaque vecteur de U.

Définition 3.12. Soit $U \subset \mathbb{R}^n$ un cône généré par l'ensemble $\{u^1, u^2, \ldots, u^k\}$. Alors, le *cône polaire semi positif* de U noté $U^{>}$ est le cône convexe :

$$U^{>} = \{ y \in \mathbb{R}^n | y^t u^i \geq 0 \text{ pour tout } i \text{ et } y^t u^i > 0 \text{ pour au moins un } i \} \cup \{0 \in \mathbb{R}^n\}$$

Notons qu'un vecteur $y \in U^{>}$ doit avoir un produit scalaire positif avec au moins l'un des u^i, $i = \overline{1,k}$. L'origine $\{0 \in \mathbb{R}^n\}$ est incluse car sinon $U^{>}$ ne serait pas un cône. [66]

3.2 Détection graphique de l'efficacité

Soit le problème de programmation linéaire multiobjectif en nombres entiers suivant : [15, 68]

$$(P) \begin{cases} "Max" \quad z^i = c^i x \quad i = \overline{1,r} \\ x \in D \end{cases}$$

$D = \{x \in \mathbb{Z}^n | Ax = b,\ x \geq 0\}$ avec : $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$ et $C = (c^i)_{i=\overline{1,r}} \in \mathbb{Z}^{r \times n}$.
Pour tester l'efficacité en un point $x^* \in D$, Ralph E. Steuer [66] a introduit le concept d'*ensemble dominant* qui est principalement basé sur la notion du cône étudiée précédemment.

Définition 3.13 (**Ensemble dominant**)**.** Soit $x^* \in D$ et $C^{>}$ le cône polaire semi positif du cône C généré par les gradients des r fonctions objectifs, i.e. :

$$C^{>} = \left\{y \in \mathbb{R}^n | y^t (c^i)^t \geq 0 \text{ pour tout } i \text{ et } y^t (c^i)^t > 0 \text{ pour au moins un } i\right\} \cup \{0 \in \mathbb{R}^n\}$$

On définit l'ensemble dominant noté ED_{x^*} , comme étant la somme des ensembles $\{x^*\}$ et $C^{>}$:

$$ED_{x^*} = x^* \oplus C^{>},$$

c'est à dire :

$$ED_{x^*} = \{x \in \mathbb{R}^n | x = x^* + y,\ Cy \geq 0,\ Cy \neq 0\}$$

L'ensemble dominant ED_{x^*} contient tous les points dont les vecteurs critères dominent le vecteur critère de $x^* \in D$. Notons que la somme des ensembles $\{x^*\}$ et $C^{>}$ effectue une translation du cône polaire semi positif de l'origine vers le point en question.

Théorème 3.1. *[66]*
Soit ED_{x^*} l'ensemble dominant en $x^* \in D$. Alors x^* est efficace si et seulement si : $ED_{x^*} \cap D = \{x^*\}$.

Preuve 1. $\Rightarrow$ / *Supposons que* x^* *est efficace et que* $ED_{x^*} \cap D \neq \{x^*\}$. *Alors, il existe* $\overline{x} \in ED_{x^*} \cap D$, $\overline{x} \neq x^*$. *Puisque* $\overline{x} \in ED_{x^*}$, *alors* $\overline{x} = x^* + y$ *où* $y \in C^>$. *Comme* $Cy \geq 0$, $Cy \neq 0$ *alors* $C\overline{x} \geq Cx^*$, $C\overline{x} \neq Cx^*$. *Ceci contredit le fait que* x^* *est efficace. Alors si* x^* *est efficace,* $ED_{x^*} \cap D = \{x^*\}$.
$\Leftarrow$ / *Si* $ED_{x^*} \cap D = \{x^*\}$, *ceci implique que si le vecteur critère de* $\overline{x}$ *domine le vecteur critère de* x^*, $\overline{x} \notin D$ *alors le vecteur critère de* x^* *est non dominé, et par conséquent,* x^* *est efficace.*

Corollaire 1. *Si* $C^{\geq} = \{0_{\mathbb{R}^n}\}$, *alors* $\forall x \in D$, x *est solution efficace.*

3.2.1 Solutions supportées et solutions non supportées

Cette section est rédigée sur la base des documents cités en [15, 68, 71].
Sur le front Pareto, deux types de solutions peuvent être différenciées : les solutions supportées et les solutions non supportées. Les premières sont celles situées sur l'enveloppe convexe de l'ensemble des solutions (Figure 3.3) et peuvent donc être trouvées à l'aide d'une agrégation linéaire des objectifs [34]. Elles sont donc plus simples à obtenir que les solutions non supportées. D'ailleurs, la principale difficulté réside dans la détermination des solutions non supportées puisqu'on ne dispose pas de méthode algorithmiquement simple pour les calculer, comme c'est le cas pour les solutions supportées et les premiers travaux en optimisation combinatoire multiobjectif se sont pour la plupart focalisés sur la recherche de ces solutions supportées en optimisant des combinaisons linéaires des objectifs utilisant différents vecteurs de poids.

3.2.2 Fonctions scalarisantes

S'il apparaît naturel de limiter la considérations des solutions efficaces comme solutions potentielles de compromis, ceci ne résout cependant pas le problème de décision proprement dit qui nécessite de sélectionner une seule solution de "meilleur compromis". cette section exige une information supplémentaire relative à la structure de préférence du décideur, elle peut parfois se traduire en termes de paramètres de préférences, appelés les poids $\lambda_i (i = 1, ..., k)$ qui reflètent l'importance relative de chaque critère (le plus souvent ils sont normés : $\sum_{i=1}^{k} \lambda_i = 1, \lambda_i \geq 0$).

Les poids, servent en général à agréger les différents critères en une fonction unique. Ces fonctions d'agrégation sont appelées fonctions scalarisantes. Elles ont le plus souvent un rôle technique, interne à la méthode ; leur optimisation (moncritère donc) permet de générer une solution efficace.

Caractérisation des solutions efficaces [34]

Les fonctions scalarisantes ci-dessus permettent de caractériser, entièrement ou partiellement, l'ensemble des solutions efficaces. Ce sont davantage des caractérisations théoriques car leur mise en œuvre conduit à la résolution de problèmes multi-paramétrique, peu aisé à traiter si le nombre de paramètres (égal au nombre de critères) est élevé. Posons

$$\Lambda = \{\lambda \in \mathbb{R}^k | \sum_{i=1}^{k} \lambda_i = 1, \lambda_i > 0, i = 1, ..., k\},$$

Théorème 3.2. *[68]*

Soient P un MOP ayant k fonctions objectifs et $\lambda \in \mathbb{R}^k$. Le problème pondéré P_λ est défini par :

$$(P_\lambda) : Max\{\lambda^t Cx \mid x \in D\}$$

avec $\lambda \in \Lambda$ et $Z_D = \{z(x) \in \mathbb{R}^k | x \in D\}$

a) Si x est une solution optimale de (P), x est une solution efficace.

b) Si x est solution efficace et que Z_D est un ensemble convexe, il existe $\lambda \in \Lambda$ tel que x est solution optimale de (P_λ).

Définition 3.14. L'ensemble des solutions efficaces est noté X_E et l'ensemble des points non dominés est Y_N.

Définition 3.15. Cette méthode de calcul, populaire grâce à sa simplicité de mise en œuvre, a conduit à la distinction des solutions efficaces et des points non dominés en deux catégories :

- les points non dominés supportés, se trouvant sur la frontière de l'enveloppe convexe de Y_N. Les ensembles de points non-dominés supportés et des solutions efficaces supportées correspondantes sont notés Y_{SN} et X_{SE}. Ces points sont obtenus par la résolution de sommes pondérées.

- les points non dominés non supportés, constitués des points de Y_N situés à l'intérieur de

son enveloppe convexe. Les ensembles de points non dominés non supportés et des solutions efficaces non supportées correspondantes sont notés Y_{NN} et X_{NE}.

Par ailleurs, on peut encore distinguer deux types de solutions parmi X_{SE} :

- les solutions efficaces supportées extrêmes dont l'image se situe sur un sommet de conv(Y). Ces solutions forment X_{SE_1} et $Y_{SN_1} = z(X_{SE_1})$ est l'ensemble des points non dominés supportés extrêmes ;

- les solutions efficaces supportées non extrêmes dont l'image n'est pas sur un sommet de conv(Y). Ces solutions forment X_{SE_2} et $Y_{SN_2} = z(X_{SE_2})$ est l'ensemble des points non dominés supportés non extrêmes.

Remarque 3.1. Alors pourquoi ne pas se satisfaire des solutions supportées ? Tout d'abord parce que ces solutions peuvent ne représenter qu'un petit sous ensemble des solutions efficaces. De plus, ces solutions supportées ne sont pas forcément bien réparties le long du front et ne représentent pas toujours un bon compromis.

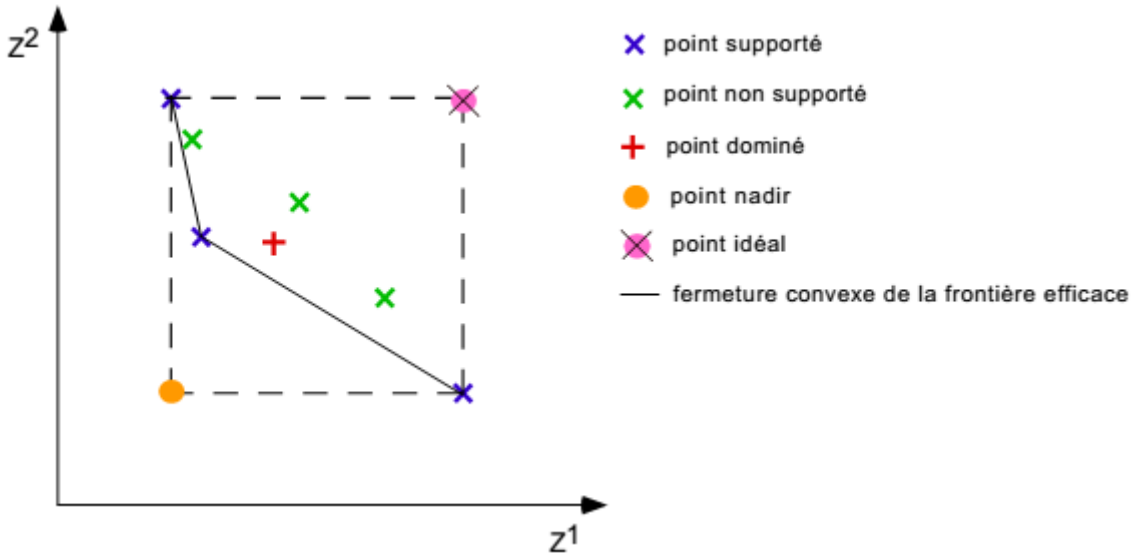


FIGURE 3.3 – Schéma des différentes solutions et points caractéristiques de l'espace des objectifs

Exemple 3.1. [71]

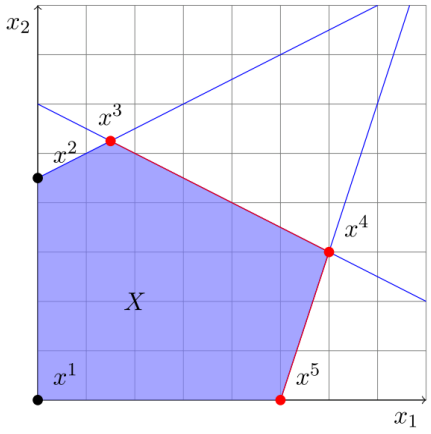
Soit le programme linéaire biobjectif suivant :

$$\begin{cases} \min z_1 = - x_1 - 4x_2 \\ \min z_2 = - 2x_1 + 2x_2 \\ s.c. \quad - x_1 + 2x_2 \leq 9 \\ \qquad\quad 6x_1 - 2x_2 \leq 30 \\ \qquad\quad x_1 + 2x_2 \leq 12 \\ \qquad\quad x_i \geq 0 \quad i = 1, 2 \end{cases} \tag{MOPL1}$$

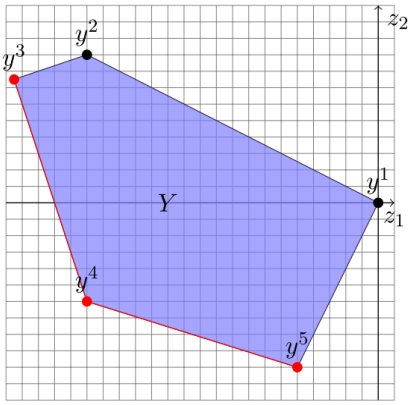
Les ensembles réalisables et efficaces de $MOPL1$ sont représentés sur les figures 3.4 (a) et 3.4 (b). En particulier, le polytope X est délimité par cinq solutions extrêmes :

- $x_1 = (0, 0)$ dont l'image est $z_1 = (0, 0)$;
- $x_2 = (0, 4.5)$ dont l'image est $y_2 = (-18, 9)$;
- $x_3 = (1.5, 5.25)$ dont l'image est $y_3 = (-22.5, 7.5)$;
- $x_4 = (6, 3)$ dont l'image est $y_4 = (-18, -6)$;
- $x_5 = (5, 0)$ dont l'image est $y_5 = (-5, -10)$.

Ici, une infinité de solutions sont efficaces : les segments $[x_3, x_4]$ et $[x_4, x_5]$. On peut remarquer que ces solutions peuvent être décrites en utilisant uniquement les trois solutions x_3, x_4,x_5. On dit que ces solutions sont efficaces extrêmes. Généralement, elles sont utilisées pour décrire implicitement l'ensemble efficace et l'ensemble non dominé, et leur nombre peut augmenter exponentiellement avec la taille du problème [28].



(a) Ensemble réalisable de (MOPL1) et solutions efficaces.



(b) Image de l'ensemble réalisable de (MOPL1) et points non dominés.

FIGURE 3.4 – Ensemble réalisable du problème $(MOIPL1)$ dans l'espace des décisions et l'espace des objectifs de l'exemple 3.1

Théorème 3.3. *[71]*

Toutes les solutions efficaces d'un $(MOLP)$ sont supportées.

Dans l'exemple 3.1 on a donc $X_E = X_{SE} = [x_3, x_4] \cup [x_4, x_5]$, $X_{SE_1} = \{x_3, x_4, x_5\}$ et $X_{SE_2} =]x_3, x_4[\cup]x_4, x_5[$. Afin d'illustrer les autres ensembles introduits, considérons un exemple de programme discret.

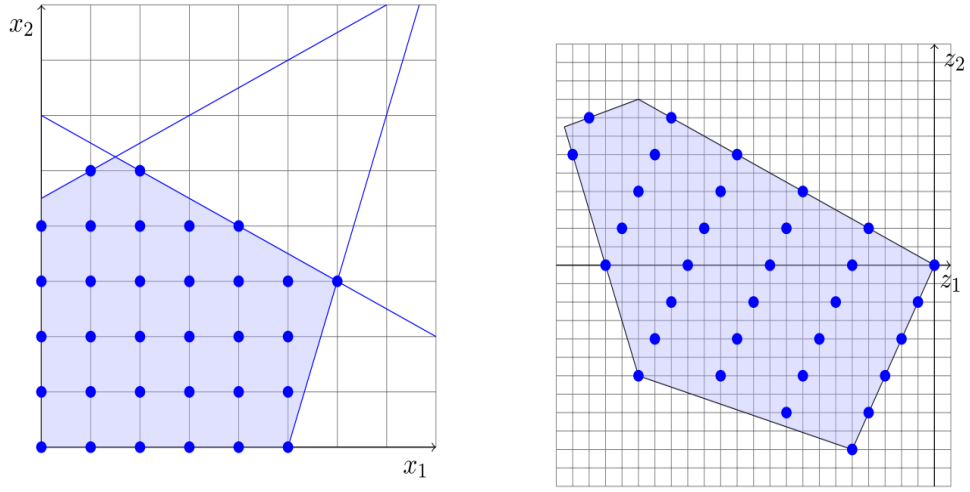
Exemple 3.2. [71] Soit le programme linéaire biobjectif suivant :

$$\begin{cases} \min z_1 = & -x_1 - 4x_2 & \\ \min z_2 = & -2x_1 + 2x_2 & \\ s.c. & -x_1 + 2x_2 & \leq 9 \\ & 6x1 - 2x_2 & \leq 30 \\ & x_1 + 2x_2 & \leq 12 \\ & x_i \in \mathbb{N} & i = 1, 2 \end{cases}$$

Les ensembles réalisable et efficace de $(MOIPL1)$ sont représentés sur les figures 3.5 (a) et 3.5 (b). Dans cet exemple :

- $X_E = \{(2,5), (4,4), (6,3), (5,0), (5,1)\}$ et $Y_N = \{(-22,6), (-20,0), (-18,-6), (-5,-10), (-9,-8)\}$
- $X_{SE} = \{(2,5), (4,4), (6,3), (5,0)\}$ et $Y_{SN} = \{(-22,6), (-20,0), (-18,-6), (-5,-10)\}$

- $X_{NE} = \{(5,1)\}$ et $Y_{NN} = \{(-9,-8)\}$
- $X_{SE_1} = \{(2,5),(6,3),(5,0)\}$ et $Y_{SN_1} = \{(-22,6),(-18,-6),(-5,-10)\}$
- $X_{SE_2} = \{(4,4)\}$ et $Y_{SN_2} = \{(-20,0)\}$



(a) Ensemble réalisable de (MOIPL1)

(b) Comparaison des valeurs des solutions extrêmes de (MOIPL1)

FIGURE 3.5 – Ensemble réalisable du problème $(MOIPL1)$ dans l'espace des décisions et l'espace des objectifs de l'exemple 3.2

Il existe donc des solutions non supportées dans les $(MOILP)$.

Exemple 3.3. [70] Considérons l'exemple suivant :

$$\begin{cases} maxz_1 = 6x_1 + 3x_2 + x_3 \\ maxz_2 = x_1 + 3x_2 + 6x_3 \\ sc. \quad 1 + x_2 + x_3 \leq 1 \\ x_j = (0,1) \qquad\qquad\qquad j = 1,2,3. \end{cases}$$

Par application du théorème de Geoffrion, nous savons que les solutions optimales du

problème paramétrique :

$$\begin{cases} \max \lambda(6x_1 + 3x_2 + x_3) + (1-\lambda)(x_1 + 3x_2 + 6x_3) \\ \text{soumis à :} \\ x_1 + x_2 + x_3 \leq 1 \\ x_j = (0,1) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j = 1,2,3. \\ \text{avec } 0 < \lambda < 1 \end{cases}$$

sont des solutions efficaces. Ces solutions optimales sont :

$$\begin{bmatrix} x_1 = 1,\ x_2 = 0,\ x_3 = 0 \Rightarrow z_1 = 6; z_2 = 1 \\ x_1 = 0,\ x_2 = 0,\ x_3 = 1 \Rightarrow z_1 = 1; z_2 = 6 \end{bmatrix}$$

Or il est aisé de constater que la solution :

$$x_1 = 0, x_2 = 1, x_3 = 0 \Rightarrow z_1 = 3; z_2 = 3$$

est également efficace mais n'est pas solution optimale du problème paramétrique (figure3.6).

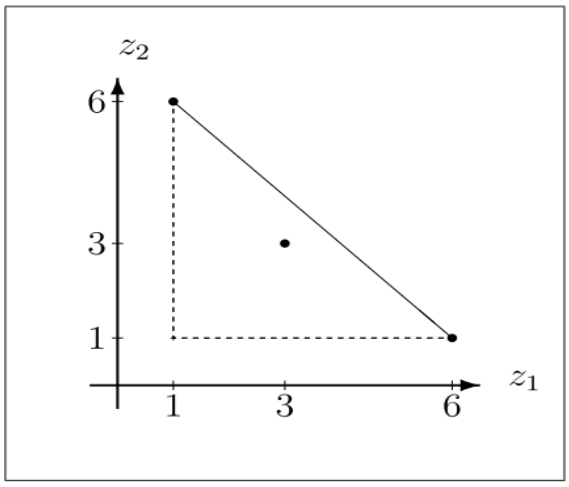


FIGURE 3.6 – Solutions efficaces supportées et non supportées

La raison en est que l'ensemble des solutions admissibles n'est pas un ensemble convexe. Nous appellerons ensemble des solutions supportées, noté **SE(P)**, l'ensemble des solutions efficaces générées par résolution du problème :

$$\max_{z \in Z_D} s_1(z, \lambda)$$

et l'ensemble de solutions non supportées, l'ensemble **E(P) \ SE(P)**. La principale difficulté théorique des problèmes $MOILP$ concerne la génération des solutions non supportées, c'est-à-dire celles qui ne sont pas sur la frontière efficace mais sont situées dans la région hachurée de la figure 3.7.

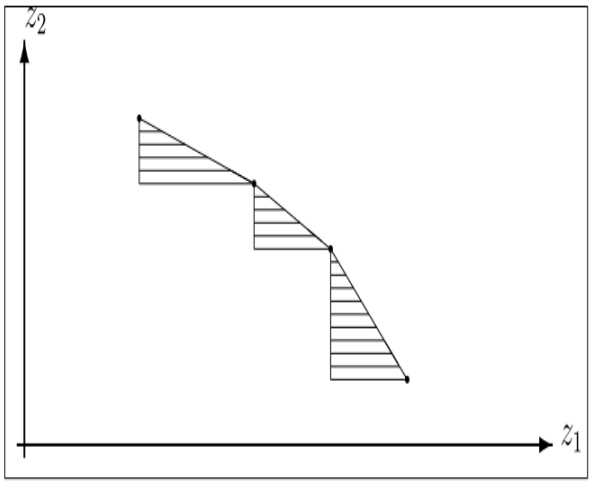


FIGURE 3.7 – Régions des solutions efficaces non supportées

3.2.3 Approches de résolution

La solution d'un problème multiobjectif est un ensemble de solutions. Cependant, pour un problème réel, une seule solution pourra être déployée. Un choix par un décideur doit donc être effectué ; le décideur peut intervenir en amont de la résolution, après celle-ci, ou de manière interactive. On distingue classiquement trois grandes familles de méthodes selon la manière dont sont combinés ces processus :

- Préférence à priori : le décideur définit ses préférences entre les différents objectifs avant d'utiliser la méthode d'optimisation.
- Préférence progressive interactive : le décideur affine son choix de compromis au fur et à mesure du déroulement de la méthode d'optimisation.
- Préférence à posteriori : le décideur choisit la solution de son choix parmi l'ensemble des solutions fournies par la méthode d'optimisation.

Il est à noter que certaines méthodes n'entrent pas forcément dans une seule catégorie. En effet, la méthode qui consiste à agréger les différents objectifs à l'aide de poids est une méthode à préférence à priori ; le décideur affectant les poids de manière à favoriser tel ou tel objectif. Cependant, si les poids sont affectés aléatoirement et si la méthode est itérée en changeant les poids à chaque exécution, il s'agit alors d'une méthode à préférence à posteriori.

3.2.4 Quelques méthodes exactes de résolution

Un grand nombre d'approches existent pour résoudre les problèmes multiobjectif. Certaines utilisent des connaissances du problème pour fixer des préférences sur les critères et ainsi contourner l'aspect multicritère du problème. D'autres mettent tous les critères au même niveau d'importance, mais là aussi il existe plusieurs façons de réaliser une telle opération.

Nous nous placerons dans le cadre de méthodes où la modélisation des préférences n'est pas requise et où le procédé d'optimisation doit être puissant afin de fournir l'ensemble des solutions de bons compromis par des méthodes exactes.

Méthode de Abbas, Chergui et Ait Mehdi [1]

L'approche [1] génère toutes les solutions entières non dominées sans passer en revu toutes les solutions possibles. C'est une méthode Branch and Bound qui fait appel aux coupes efficaces pour passer d'un vecteur entier à un autre. Elle fait appel à la méthode du simplexe pour résoudre le programme linéaire P_l à l'étape l de la méthode. Les vecteurs critères sont alors adjoints au tableau du simplexe et évoluent de façon dynamique dans ce dernier de la même façon que le critère z de P_l. Si la solution optimale obtenue de P_l n'est pas entière, ce qui correspond à un nœud de type 1 de l'arbre, seul un processus de branchement est effectué pour détecter une solution entière (contrairement aux autres méthodes existantes, la méthode n'a pas besoin de rechercher une solution entière optimale, mais seulement une solution entière voisine à la solution optimale courante qui n'est pas entière). Quand une solution entière est obtenue, le nœud est de type 2, le vecteur critère correspondant est comparé à ceux déjà trouvés et l'ensemble des solutions potentiellement non dominées est mis à jour. En vue de rechercher une autre solution entière, les directions

d'amélioration des critères sont utilisées pour construire des coupes efficaces permettant d'éviter l'exploration de domaines ne contenant que des solutions entières dominées.

Notations

$$(P_l)\begin{cases} Max\ Z = \sum_{i=1}^{k} c^i x \\ x \in X_l \end{cases} \tag{3.2.1}$$

Où $X_0 = X$ et $X_{l+1} = \cup_{i \in K_l}\{x \in X_l | c^i x \geq c^i x^l + \sum_{j\ inN_l \setminus H_l^i} \lfloor \widehat{c_j^i} \rfloor x_j + max\{1, \lfloor \widehat{c_{j0}^i} \rfloor\}\}$

- x^l : Une solution du problème (P_l).
- B^l : Ensemble des indices des variables de base de x^l.
- N^l : Ensemble des indices des variables hors base de x^l.
- $\hat{c}^i$: Vecteur critère réduit relatif au critère $i, i = \{1, ..., k\}$.
- $\hat{c}_j^i$: La $j^{ième}$ composante du coût réduit du vecteur critère i.
- H_l^i : L'ensemble définit comme suit : $H_l^i = \{j \in N^l \mid \hat{c}_j^i > 0\}$
- K_l : $K_l = \{i \in \{1, ..., k\} \mid H_l^i \neq \emptyset\}$

  est l'ensemble des critères pouvant être améliorés à partir de x^l.
- Pour chaque $i \in K_l$, on définit la coupe efficace suivante :

$$c^i x \geq c^i x^l + \sum_{j \in N^l \setminus H_l^i} \lfloor \hat{c}_{j_0} \rfloor x_j + max\{1, \lfloor \hat{c}_j^i \rfloor\}. \tag{3.2.2}$$

  où : $\hat{c}_{j_0} = \min_{j \in H_l}\{\hat{c}_j^i\}$ pour $i \in K_l$.

Algorithme 1. *$SND = \emptyset$, $i = 0$, $pg = \{(P_0)\}$.*

1. *Tant que $pg \neq \emptyset$*
   - *Choisir un programme (P_l) de pg et $pg = pg/(P_l)$.*
   - *Résoudre (P_l).*
   - *Si (P_l) est irréalisable,$i := i + 1$.*
   - *Sinon,soit x^l la solution trouvée, aller à 2*

   Fin tant que
2. - *Si x^l est entière.*
     - *Si Cx^l n'est pas dominée par toute solution de SND alors $SND := SND \cup$*

$\{Cx^l\}$.

- *Si* Cx^l *domine* Cy *,une solution de* SND*,alors* $SND := SND \setminus \{Cy\} \cup \{Cx^l\}$.
- *Déterminer :* N^l , H_l^j *pour chaque vecteur critère* j *et* K_l.
- *Si* $|K_l| = \emptyset$*,aller à 1*
- *Sinon,*$|K_l|$ *nouveaux programmes ajoutée au pg :*

$$\begin{cases} (P_l) \\ \\ \text{la contrainte } 3.2.2 \end{cases}$$

aller à 1

- *Sinon, deux nouveaux programmes ajoutée au pg :*

$$\begin{cases} (P_l) \\ \\ x \leq \lfloor x_j \rfloor \end{cases} \qquad \begin{cases} (P_l) \\ \\ x \geq \lceil x_j \rceil \end{cases}$$

aller à 1

Méthode ϵ-contrainte. [39]

La méthode a été proposé par Haïmes et *al.* (1971). Elle consiste à transformer les $(k-1)$ objectif en contraintes. L'objectif restant, qui peut être choisi arbitrairement, est la fonction d'objectif du problème d'optimisation mono-objectif qui en résulte, en d'autre termes dans cette approche, le problème consiste à optimiser une fonction f_i sujette à des contraintes sur les autres fonctions. En effet, en utilisant cette méthode itérativement, en repartant à chaque fois de la solution trouvée pour définir la borne suivante [39], il est possible en utilisant une méthode exacte mono-objectif de générer l'ensemble des solutions Pareto.

Soit le problème à résoudre :

$$(P) \begin{cases} min & f(x) = (f_1(x), f_2(x), ..., f_k(x)) \\ \\ sous & x \in S \end{cases} \tag{3.2.3}$$

où $f : \mathbb{R}^n \to \mathbb{R}^k$ telle que $f(x) = (f_i(x))_{i=1,...,k}$

Pour un objectif $i \in \{1, ..., k\}$ du problème (P), résoudre le problème suivant :

$$(P_i)\begin{cases} min & f_i(x) \\ x \in S & \\ f_j(x) \leq \epsilon_j & \forall j \in \{1,...,k\}\backslash\{i\} \end{cases} \tag{3.2.4}$$

où ϵ_j est une borne supérieure pour le $j^{ième}$ objectif.

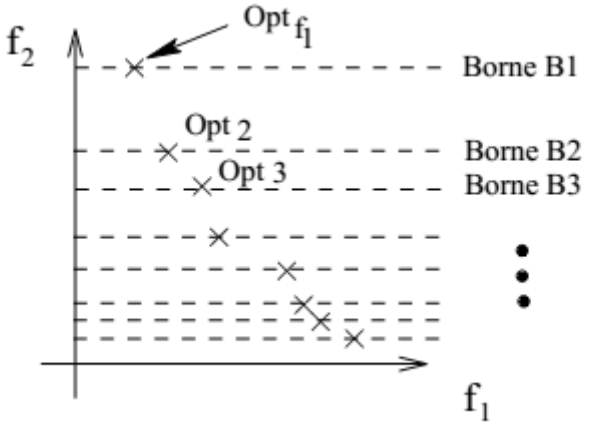


FIGURE 3.8 – Illustration de la méthode $\epsilon-$contrainte.

La figure 3.8 illustre un exemple pour lequel, la solution efficace optimale pour l'objectif $f1$ est d'abord recherchée (solution Opt_{f1}). Cette solution détermine la borne $B1$ sur l'objectif $f2$ en dessous de laquelle l'objectif $f2$ va devoir être optimisé. Cela nous donne la solution Opt_{f2} qui elle même détermine la borne $B2$, etc...

La méthode Epsilon-Contrainte est facile à utiliser. Elle permet de trouver les solutions du front même les fronts non-convexe. L'inconvénient principal de cette méthode est de trouver les bornes de variation pour les fonctions objectifs qui seront transformées en contraintes.

Méthode de Özlen et Azizoğlu [50]

La méthode de Özlen et Azizoğlu est une amélioration de la méthode $\epsilon-$contrainte. Dans la méthode [39], les valeurs ϵ_j sont diminuées d'une unité dans le cas de minimisation multiobjectif, la méthode décrite dans [50] diminue ces valeurs en tenant compte de la solution non dominée actuelle. En outre, la fonction objectif n'est plus l'un des critères, mais une combinaison pondérée et appropriée des critères de (P) qui assure l'efficacité des

solutions obtenues.

Dans cette méthode, on montre que le problème multiobjectif (P) est équivalent (au sens des solutions efficaces) au problème suivant, obtenu en transformant le $k^{ième}$ critère en contrainte :

$$\begin{cases} min & f_1(x) + \epsilon_k f_k(x) \\ \vdots & \\ min & f_{k-1}(x) + \epsilon_k f_k(x) \\ f_k(x) \leq l_r & \\ x \in S & \end{cases} \tag{3.2.5}$$

où l_k est une borne supérieure de f_k. Ce problème est à son tour équivalent au problème obtenu en transformant le critère $k-1$ en contrainte et ainsi de suite jusqu'à arriver à la résolution du programme linéaire en nombres entiers mono-objectif suivant :

$$\begin{cases} min & f_1(x) + \sum_{i=2}^{k} \omega_i f_i(x) \\ f_i(x) \leq l_i & i = 2, ..., k \\ x \in S & \end{cases} \tag{3.2.6}$$

où les poids ω_i sont calculés en fonction des $\overline{f_i}$ et $\underline{f_i}$, bornes supérieure et inférieure de f_i respectivement, de sorte que la solution optimale de 3.2.6 soit efficace pour le problème (P) et induit donc, une solution non dominée.

L'idée est que pour résoudre un problème bi-objectif, on le ramène à un problème mono-objectif. Pour résoudre un problème tri-objectif, on détermine d'abord les solutions non dominées d'un problème biobjectif correspondant, et de façon générale, pour résoudre un problème avec r objectifs, on doit le ramener à un problème à r, on doit le ramener à un problème à on doit le ramener à un problème à $r-1$ objectifs en utilisant chaque fois la transformation $\epsilon-$contrainte.

Méthode récursive améliorée de Özlen et Burton [51]

La principale contribution est d'améliorer l'algorithme récursif d'Özlen et Azizoğlu pour générer l'ensemble non-dominé en utilisant l'ensemble de sous-problèmes déjà résolus

et leurs solutions pour éviter de résoudre un grand nombre de programmes en nombres entiers (IP).
Un inconvénient majeur de l'algorithme récursif décrit dans [50] est qu'il ne stocke pas ou n'utilise pas de sous-problèmes d'information qui ont déjà été résolus. Dans cet esprit, les auteurs dans [51] proposent un algorithme amélioré qui utilise ces informations pour éviter de résoudre un grand nombre d'IP.

Méthode en deux phases [69]

L'algorithme en deux phases est un cadre méthodologique pour les MOCO proposée par Ulungu et Teghem [69]. L'idée générale est de construire l'ensemble des solutions efficaces en séparant la recherche des solutions supportées et la recherche des solutions non supportées, et de réutiliser des algorithmes efficaces pour le cas mono-objectif quand il en existe. Ces algorithmes ayant été conçus pour profiter pleinement de la structure du problème, toute modification de cette structure empêche leur utilisation. C'est pourquoi on s'interdit avec cette méthode d'ajouter des contraintes sur les valeurs des fonctions objectifs afin de rechercher les solutions efficaces. L'objectif de la première phase est d'obtenir l'ensemble des solutions Pareto supportées. Comme nous l'avons vu précédemment, ces solutions ont l'avantage d'être relativement faciles à trouver puisqu'elles optimisent une certaine combinaison linéaire des objectifs. Ainsi, durant la première phase de la méthode, les deux solutions extrêmes (solutions optimisant chacun un des deux objectifs) sont recherchées. Puis, de façon récursive, dès que deux solutions supportées sont trouvées, la méthode recherche d'éventuelles autres solutions supportées entre les deux premières à l'aide de combinaisons linéaires bien choisies des objectifs. A la fin de la première phase l'ensemble des solutions supportées est donc trouvées solutions supportées dans l'intervalle. Ensuite, les autres solutions efficaces sont calculées durant la phase 2. Cette phase tire partie des solutions obtenues lors de la phase 1 pour déterminer des zones de recherche dans lesquelles des points non dominés peuvent être trouvés. Ces zones de recherche sont ensuite visitées au moyen d'une stratégie énumérative garantissant la détermination d'un ensemble complet.

La méthode en deux phases présente un schéma de résolution exacte très intéressant car très général et qui ne dépend pas du problème. Son intérêt réside dans une décomposition de l'espace de recherche et l'utilisation de méthodes mono-objectif pour les différentes

résolutions successives (recherche des points extrêmes, résolution par agrégation...). Appliquer la méthode en deux phases pour la résolution d'un problème biobjectif nécessite donc d'avoir une méthode mono-objectif efficace (si possible polynomiale). C'est le cas pour les problèmes traités par Ulungu et Teghem, et c'est ce qui rend leur méthode performante pour ces problèmes là. Cependant, lorsque la restriction du problème à un seul objectif génère un problème déjà NP-difficile, la non existence de méthode exacte efficace pouvant optimiser chaque objectif séparément peut compromettre l'intérêt de la méthode et notamment rendre la première phase très coûteuse. Dans le cas biobjectif, la zone de recherche est donnée par l'ensemble des triangles dont les hypoténuses sont définies par des points supportés adjacents (c'est-à-dire consécutifs par rapport à un objectif). En général, ces triangles sont explorés séparément par des techniques énumératives en utilisant des bornes inférieures et supérieures, des coûts réduits... Contrairement à la première phase, l'exploration de la zone de recherche doit utiliser les spécificités du problème pour être efficace.

Chapitre 4

Problème de l'arbre multiobjectif (MOST) - État de l'art

Le problème de l'arbre à objectifs multiples ($MOST$) se pose dans beaucoup de contextes. Comme exemple, nous citons celui de l'optimisation de la qualité de service dans un réseau de télécommunication dont l'évaluation est exprimée selon un ensemble de critères : délai minimum, débit maximum, taux d'erreurs minimum, coût d'utilisation minimum,...etc. Dans notre étude, nous nous intéressons à la recherche l'ensemble de tous les arbres efficaces d'un graphe connexe dont chacune des arêtes est munie d'un vecteur de poids de dimension $p \geq 2$. Ce problème est connu pour être de type NP-difficile même pour $p = 2$, [10]. À notre connaissance, peu d'approches de résolution ont été implémentées, particulièrement pour des problèmes comportant plus de deux objectifs.

4.1 Position du problème

La résolution d'un problème multiobjectif consiste à déterminer soit l'ensemble des arbres efficaces dans l'espace des décisions, soit l'ensemble des arbres non dominées dans l'espace des critères.

4.1.1 Définitions et notations

Étant donnée un graphe $G = (V, E)$ d'ordre n, non orienté, simple et connexe, dans lequel $V = \{v_1, v_2, \ldots, v_n\}$ représente l'ensemble des sommets et $E = \{e_1, e_2, \ldots, e_m\}$ est

l'ensemble des arêtes et chaque arête $e_i \in E, i = \overline{1,m}$, est valuée par un vecteur-coût $c_i = (c_{ik}),\ k = \overline{1,r}, r \geq 2$. Soit $T \subset E$ un arbre de G, le vecteur-coût de T est donné par $C_k(T) = \sum_{e_i \in T} c_{ik},\quad k = \overline{1,r}$. On note $C(T) = (C_k(T))_{k=\overline{1,r}}$. (voir [17]).

Définition 4.1. On dit qu'un vecteur $C(T)$ domine un autre vecteur $C(T')$ si $C_k(T) \leq C_k(T')\ \ \forall k = \overline{1,r}$ avec $C_k(T) < C_k(T')$ pour au moins un indice $k \in \{1, ..., r\}$.

Définition 4.2. L'arbre T de G est efficace s'il n'existe aucun arbre T' de G tel que $C(T')$ domine $C(T)$.

On rappelle que dans la programmation multiobjectif, le point idéal est obtenu en collectant chacune des valeurs objectives optimales. [66].

Définition 4.3. Pour le problème $MOST$, le point idéal I a pour coordonnées I_k tel que :

$$I_k = C_k(T) = \min \{C_k(T) : T \textit{ arbre de } G\}, k = \overline{1,r}.$$

Définition 4.4. Les arêtes de G de type $e2c$ sont des arêtes qui appartiennent à au moins deux cycles de G.

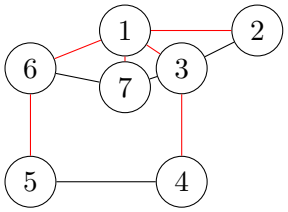
4.1.2 Base de cycles

L'étude des bases du cycle remonte aux premiers jours de la théorie des graphes ; MacLane (1937) [45] a donné une caractérisation des graphes planaires en termes de bases de cycles. Pour plus de détails, voir [45, 49].

On note l'ensemble $\overline{T} = \{e \in E : e \notin T\}$. Alors, pour chaque arête $e \in \overline{T}$, $T \cup \{e\}$ contient un unique cycle μ_e et $B = \{\mu_e, e \in \overline{T}\}$ est une **base de cycle** de G. Ce qui signifie que le vecteur représentatif $V(\mu_e)$ des arêtes des cycles $\mu_e \in B$ forment une base du sous espace vectoriel de dimension $(m - n + 1)$ de R^m, où les coordonnées de $V(\mu_e)$ sont données par $V_j(\mu_e) = 1$ si l'arête $e_j \in \mu_e$ et 0 sinon, pour tous $j = \overline{1,m}$.

Théorème 4.1. *Soit G un graphe connexe, T un arbre de G ; e est une arête de G ne figurant pas dans T, son adjonction à T détermine un cycle μ_e d'après la propriété (4) du théorème 1.4, et les différents cycles μ_e constituent une base de cycles indépendants (appelés les cycles associés a l'arbre T) [4].*

Exemple 4.1. Soit le graphe G connexe, simple et non orienté avec 7 sommets et 10 arêtes, dont les arête coloriées en rouge forment un arbre couvrent T de G.



Considérons un arbre T de G défini par ses arêtes $\{12, 13, 16, 17, 34, 65\}$ et le coarbre associé $\overline{T} = \{23, 37, 45, 67\}$. Le nombre cyclomatique de $G = m - n + 1 = 10 - 7 + 1 = 4$, donc les 4 cycles indépendants sont :

$T \cup \{23\} \rightarrow \mu_1 = \{12, 13, 23\}$.

$T \cup \{37\} \rightarrow \mu_2 = \{13, 17, 37\}$.

$T \cup \{45\} \rightarrow \mu_3 = \{13, 16, 34, 65, 45, \}$.

$T \cup \{76\} \rightarrow \mu_4 = \{13, 16, 17, 67\}$.

4.1.3 Modèle mathématique

Un modèle mathématique associé au problème $MOST$ est donné comme suit [40] :

$$x_i = \begin{cases} 1 & \text{si l'arête } \ e_i \in T, \quad i = \overline{1,m} \\ 0 & \text{sinon} \end{cases}$$

$$(P) \begin{cases} MinZ_1 = \sum_{i=1}^{m} c_{i1} x_i \\ MinZ_2 = \sum_{i=1}^{m} c_{i2} x_i \\ \qquad \vdots \\ MinZ_r = \sum_{i=1}^{m} c_{ir} x_i \\ \sum_{i=1}^{m} x_i = n - 1 \qquad \text{.....} \quad (1) \\ \sum_{e_i \in E(S)} x_i \leq |S| - 1, \forall S \subset V, S \neq \emptyset \quad \text{...} \quad (2) \\ x_i \in \{0, 1\} \quad \forall i = \overline{1,m} \end{cases}$$

où $E(S)$ représente l'ensemble des arêtes du sous-graphe induit par S, $S \subset V$.

On note que, dans ce modèle (P), le nombre de contraintes de type (2) (les inégalités éliminant la création des cycles) est exponentiel et toute tentative de résolution de ce modèle par une méthode s'avère inutile déjà dans le cas monobjectif.

Théorème 4.2. *(Hamacher and Ruhe [40]) Un arbre efficace extrême est la solution du programme paramétrique suivant :*

$$(P_\lambda)\begin{cases} min(\lambda, T) = \sum_{i=1}^{r} \lambda_i Z_i(T) \\ \lambda \in \tau \\ \lambda_1 + \lambda_2 ... + \lambda_i = 1 \\ \lambda_i > 0, \quad \forall i = 1...r \end{cases} \tag{4.1.1}$$

Un problème d'optimisation combinatoire multiobjectif est dit intraitable si le cardinal de l'ensemble des solutions non dominées, peut être exponentiel relativement à la taille de l'instance (voir [28]). $MOST$ est intraitable et NP-difficile, même pour $r = 2$ fonctions objectifs.

Théorème 4.3. ***(Camerini, Galbiati, and Maffioli [10])*** *Le problème MOST est "NP-difficile".*

Théorème 4.4. ***Hamacher and Ruhe [40]*** *Le problème MOST est intraitable.*

Les preuves des théorèmes 4.2 et 4.4 sont données dans l'article [40].

4.2 État de l'art

En optimisation monocritère, les algorithmes de Prim (Prim, 1957) et Kruskal (Kruskal, 1956) permettent de déterminer l'arbre de poids minimal en temps polynomial. Les extensions multicritères de ces algorithmes pour le problème $MOST$ ont rencontré des difficultés inattendues, entraînant la proposition de plusieurs algorithmes inexacts dans la littérature. Nous présentons ici brièvement les raisonnements qui ont mené à ces erreurs.

4.2.1 Extension multicritère des algorithmes "Prim" et "Kruskal"

Nous considérons dans cette sous-section des généralisations des algorithmes de Prim et de Kruskal pour le problème $MOST$ à objectifs multiples [33].

I- Extension multicritère de l'algorithme de Prim

L'algorithme de Prim appliqué à un graphe scalaire construit un arbre optimal à partir d'une solution partielle optimale en choisissant, à chaque itération, un sommet adjacent amenant à une solution partielle optimale. Une généralisation est due à Corley [17], qui suggère de composer de façon itérative des arbres similaires à l'idée de Prim. Dans chaque itération, l'ensemble des sommets déjà contenus dans le sous-arbre définit une coupe dans le graphe. Le sous-arbre est agrandi par toutes les arêtes non dominées le long de cette coupe. Contrairement au cas monobjectif, un ensemble d'extensions efficaces doit être envisagé et, par conséquent, un ensemble de sous-arbres doit être traité. Le but est donc d'étendre l'algorithme de Prim au cas multicritère en construisant une arborescence de recherche de la manière suivante :

- la racine de l'arborescence de recherche est un sommet quelconque de V.
- un nœud de l'arborescence de recherche correspond à un graphe partiel connexe sans cycle couvrant un sous-ensemble de sommets $v_0 \subseteq V$ (i.e. ce que nous avons aussi appelé un sous-arbre).
- une branche de l'arborescence correspond au choix d'augmenter le sous-arbre par une arête efficace parmi les arêtes adjacentes à ce sous-arbre ne formant pas de cycle. Une branche est créée pour chacune de ces arêtes.

Or, l'ensemble des solutions retournée par l'algorithme de Corley est en fait un sur-ensemble de l'ensemble de Pareto. Hamacher et Ruhe (1994) ont signalé cette erreur et ont alors suggéré de couper les sous-arbres dominés à chaque niveau de profondeur de l'arborescence de recherche. Cette version modifiée de l'algorithme de Corley est reprise par Zhou et Gen [72]. Cependant, comme l'ont montré Knowles et Corne [46], des arbres efficaces peuvent être omis. Cela implique également que certains arbres non efficaces peuvent être retournés par l'algorithme et ne peuvent pas être filtrés. Pour s'en convaincre, nous reprenons le contre-exemple proposé par Spanjaard [63], dont le graphe complet K_4 de la figure ci-dessous :

Les arbres efficaces de ce graphe sont les arbres $T_1 = \{b, c, d\}, T_2 = \{c, d, e\}, T_3 = \{b, c, f\}$ et $T_4 = \{c, e, f\}$. Or, en choisissant le nœud 3 à la racine de l'arborescence, la version modifiée de l'algorithme de [17] retourne les arbres $T_5 = \{a, c, f\}, T_3$ et T_4. Les arbres T_1 et T_2 sont omis et l'arbre T_5 est retourné alors qu'il est dominé (par T_1). De plus, T_5 n'étant dominé ni par T_3 , ni par T_4, on ne peut savoir qu'il n'est pas efficace sans recourir

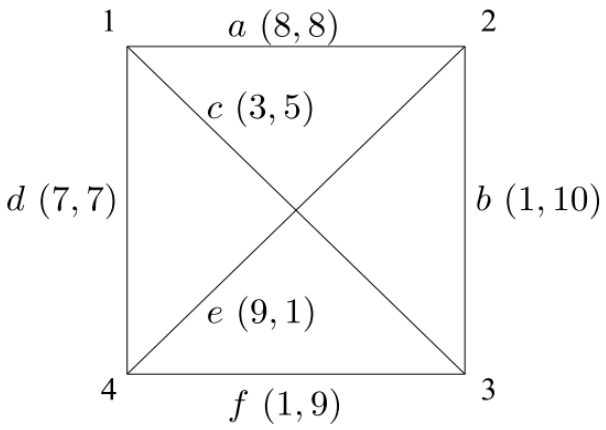


FIGURE 4.1 – Contre-exemple pour la version modifiée de l'algorithme de Corley (1985) [17]

à une énumération exhaustive de tous les arbres du graphe. Cette erreur est due au fait que l'arbre T_1 ne comporte pas de sous-arbre efficace parmi les sous-arbres à la profondeur 2 de l'arborescence (i.e.composé de deux arêtes). En effet, le sous-arbre $\{c, d\}$ de coût (10, 12) est dominé par le sous-arbre $\{e, f\}$ de coût (10, 10), et le sous-arbre $\{b, c\}$ de coût (4, 15) est dominé par le sous-arbre $\{c, f\}$ de coût (4, 14).

Selon Spanjaard [63], ces inexactitudes successives pourraient provenir d'une confusion faite lors de l'extension multiobjectif de l'algorithme de Prim entre :

1. choisir une arête telle que la solution augmentée obtenue soit efficace parmi les complétions de cette même solution partielle.

2. choisir une arête telle que la solution augmentée soit efficace parmi toutes les complétions de toutes les solutions partielles de même taille.

En effet, le premier cas revient à choisir une arête efficace parmi les arêtes restantes comme l'a proposé Corley (1985), ce qui n'est pas équivalent au deuxième cas.

II- Extension multicritère de l'algorithme de Kruskal

La généralisation provient de Serafini [60]. Son algorithme repose sur le théorème suivant :

Théorème 4.5. *Pour tout arbre efficace T, il existe un rangement des arêtes compatible avec la Pareto-dominance pour lequel l'algorithme de Kruskal retourne T.*

L'algorithme de Kruskal appliqué à un graphe scalaire construit un arbre optimal à partir d'une solution partielle optimale en choisissant, à chaque itération, l'arête optimale parmi les arêtes non comprises dans la solution partielle. Serafani (1986) a proposé

d'étendre ce principe au cas multicritère par le maintien d'un ensemble de forêts (i.e. un graphe partiel sans cycle non connexe) en procédant de la sorte :

- stocker toutes les arêtes efficaces du graphe dans l'ensemble F_1 des forêts à une arête.
- à l'itération $i(< |V|)$: pour chaque forêt $F \in F_i$ et pour chaque arête efficace $a \in A - F$ ne formant pas de cycle, construire une nouvelle forêt $F' = F \cup \{a\}$ et insérer F' dans F_{i+1}.

Cependant, l'algorithme proposé par [60] n'est pas exact non plus. En effet, d'une part, des solutions efficaces peuvent ne pas être retournées par cet algorithme, d'autre part, parmi les solutions retournées, certaines peuvent être Pareto-dominées. Si l'on applique cet algorithme sur le graphe de la figure 4.1 par exemple, l'arbre efficace T_1 ne sera pas retourné alors que l'arbre $T_6 = \{a, c, d\}$ de coût (18, 20), et donc Pareto-dominé par T_4, sera retourné.

On constate d'après ces échecs qu'il n'est pas évident d'élaborer une méthode constructive (i.e. gloutonne) pour la recherche de l'ensemble des arbres efficaces en se basant sur les algorithmes constructifs de l'optimisation monocritère. Ainsi, un des problèmes les plus faciles en optimisation monocritère devient un problème pour lequel toute intuition est faussée en optimisation multicritère.

4.2.2 Méthodes d'optimisation combinatoire multiobjectif appliquées au problème MOST

- Dans [40], les auteurs abordent la théorie du problème de l'arbre à objectifs multiples et fournissent un algorithme pour le cas biobjectif, qui se base sur la méthode en deux phases. La première phase qui permet de calculer l'ensemble des arbres efficaces supportés, est réalis´ee en se basant sur l'ordre lexicographique et l'ordre lexicographique inverse pour obtenir les arbres optimaux efficaces. Étant donnée deux arbres optimaux efficaces, un programme de somme pondérée est résolu pour déterminer soit un nouvel arbre extrême efficace entre les deux premiers ou de décider qu'il n'y a pas une telle solution. Lors de la seconde phase, une recherche de voisinage (voir définition 1.11) à partir des solutions supportées. Étant donnés deux arbres T_1 et T_2 voisins et les deux arêtes e et f les distinguant ($T_1 \setminus T_2 = \{e\}$ et $T_2 \setminus T_1 = \{f\}$), il faudrait alors ne considérer que les échanges d'arêtes entre deux voisins tels que le vecteur de coût $c(e) - c(f)$ ait des composantes strictement

négatives et strictement positives, puisque les arbres Pareto-optimaux doivent avoir des coûts incomparables. Ehrgott et Klamroth (1997) [27] ont démontré que cette méthode ne fonctionne pas. et ont exhibé un contre-exemple.

- Dans [18], les auteurs proposent de trouver l'ensemble des solutions non dominées du problème de l'arbre minimum biobjectif en le ramenant à un problème monobjectif. La méthode repose sur les fonctions somme pondérées qui fait appel à l'algorithme de Kruskal une seule fois pour obtenir le premier arbre efficace supporté T_1. Ensuite, les arêtes qui n'appartiennent pas à T_1 sont remplacées par des arêtes qui doivent quitter T_1 compte tenu de la préservation de la non-dominance, pour construire un nouvel arbre T_2 efficace.

- Une procédure en deux phases pour le problème de l'arbre minimal biobjectif est proposé par Ramos et al. [55], leur procédure ne permet pas seulement de trouver une solution efficace pour chaque point non-dominé, mais toutes les solutions efficaces. A cet effet, une procédure énumérative pour trouver tous les arbres optimaux du problème mono-objectif est développée. La première phase de l'algorithme pour le problème biobjectif est similaire à l'algorithme de [40]. Cependant, étant donné deux points extrêmes consécutifs, toutes les solutions efficaces donnant le même point non-dominé sont calculées par l'algorithme d'énumération à objectif unique. Des solutions efficaces non prises en charge sont trouvées par une méthode "Branch and Bound". Un arbre de recherche ayant N niveaux, de la racine 0 au niveau le plus bas $N-1$, est construit et passé dans la stratégie profondeur d'abord. Chaque sommet x de cet arbre de recherche correspond à une arête de la liste d'adjacence du sommet $i \in V$. De plus, avec chaque nœud x une solution partielle (une forêt) est associée. Cette solution correspond à toutes les arêtes sur le chemin de la racine de nœud de recherche au nœud x. Le branchement est effectué pour chaque arête dans la liste d'adjacence du sommet $i+1$. Les arêtes qui font déjà partie de la solution partielle ou celles qui conduisent à un cycle dans la solution partielle sont négligées. Pour chaque solution partielle, une limite inférieure du vecteur de coût est calculée. Si cette limite inférieure est dominée par une solution préalablement trouvée, la branche correspondant à cette solution est sondée. Les solutions trouvées par cette méthode "Branch and Bound" ne doivent pas nécessairement être efficaces. Ainsi, une procédure de filtrage supplémentaire doit être appliquée afin d'exclure les solutions dominées. La performance de cet algorithme en deux phases est étudiée numériquement. L'avantage c'est le temps d'exécution qui di-

minue avec un nombre croissant de sommets. Ramos et al. [55] notent que le nombre de solutions efficaces non supportées diminue pour leur configuration d'essai avec un nombre croissant de sommets.

- Andersen, Jörnsten, et Lin [2] ont contribué par une comparaison numérique de deux heuristiques pour le problème de l'arbre biobjectif. Dans les deux heuristiques, l'ensemble des solutions non dominées supportées est trouvé dans la première phase par un algorithme, par exemple, l'algorithme dans [40]. Bien que le fait que l'ensemble des solutions efficaces ne soit pas connexes, les auteurs suggèrent d'exploiter l'adjacence pour construire une approximation de l'ensemble non-dominé. Deux méthodes différentes sont proposées pour générer des arbres supportés : Une recherche par voisinage "neighborhood search" et une recherche par adjacence "adjacent search". Dans la recherche par voisinage, les arbres qui sont adjacents (au sens de [27]) à au moins un arbre efficace déjà trouvé (localement) sont étudiés. Dans la recherche par adjacence, les arbres qui sont construits sont ceux adjacents à au moins deux arbres (localement efficaces). Deux méthodes sont proposées pour générer des arbres adjacents à au moins deux arbres donnés, ces deux heuristiques sont testées sur différents cas. En moyenne, la recherche par adjacence étudie moins de points candidats que la recherche par voisinage et le temps de calcul durant la construction de nouvelles solutions candidates est cependant plus important.

- L'étude numérique de Steiner et Radzik [65] compare deux méthodes différentes en deux phases pour le problème de l'arbre minimum entier biobjectif. Pour la première phase, l'algorithme de Hamacher et Ruhe [40] est utilisé pour trouver l'ensemble des solutions efficaces supportées pour les deux approches. Dans la seconde phase, l'algorithme d'arbre minimum (mono-objectif) "k-best" de Gabow [32] est comparé à un algorithme de "Branch and Bound". Dans l'ancienne variante, les triangles entre toutes les paires de points extrêmes non dominés consécutifs sont recherchés pour des arbres efficaces non supportés par l'algorithme "k-best". Pour obtenir un arbre de coût unique, les deux composantes du coût sont agrégées par une somme pondérée. L'algorithme de Gabow [32] est alors appliqué avec cette fonction de coût. Chaque solution est utilisée pour réduire l'espace de recherche en divisant le rectangle original en deux plus petits. Cette approche "k-best" est comparée à la procédure "Branch and Bound" de Ramos [55] dans les tests numériques.
Steiner et Radzik [65] concluent que l'algorithme "k-best" est nettement plus performant

que la méthode "Branch and Bound" en raison de l'arbre de recherche énorme et de la procédure de délimitation plutôt inefficace de la seconde approche.

- Sourd et Spanjaard [62] ont proposé et mis en œvre une procédure "Branch and Bound" aussi pour le cas biobjectif, qui surpasse tous les algorithmes précédents. L'instance est pré-traitée en appliquant les règles de coloration généralisée présentées dans leurs papier, notamment grâce à une gestion fine des bornes supérieures et inférieures au cours de la recherche. Dans leur algorithme, un nœud n de l'arborescence de recherche peut être supprimé de l'arborescence s'il existe une hyper surface séparatrice entre une borne supérieure des solutions efficaces déjà connues et une borne inférieure des solutions que l'on peut obtenir à partir du noœud n. La borne inférieure qu'ils proposent d'utiliser est celle proposée par Ehrgott et Gandibleux [25] qui consiste à déterminer les solutions supportée d'un sous-ensemble de solutions à l'aide de l'optimisation de somme pondérées des coûts. Cependant, Sourd et Spanjaard [62] montrent que dans le cas bicritère, pour m arêtes il existe au plus $\frac{m(m-1)}{2}$ façons d'ordonner les m arêtes par une somme pondérée de leur coût. Ainsi, ils proposent de déterminer ces $\frac{m(m-1)}{2}$ ordres au cours d'une phase d'initialisation, puis d'appliquer l'algorithme de Kruskal sur l'ordre adéquat au cours de la recherche. Ce calcul préalable permet d'obtenir un gain de temps considérable lors de leurs expérimentations. La borne supérieure qu'ils proposent est définie par l'ensemble des points non dominés par les solutions détectées. Afin d'initialiser cet ensemble, ils déterminent au préalable un ensemble de solutions efficaces approché par la méthode de voisinage proposée par Hamacher et Ruhe [40]. Enfin, ils exploitent aussi le fait que les coûts sont entiers, améliorant ainsi l'efficacité des coupes dans l'espace de recherche.
L'algorithme de "Branch and Bound" qu'ils proposent permet de déterminer l'ensemble des solutions efficaces du problème bicritère de l'arbre allant jusqu'à 500 nœuds, ce qui, à notre connaissance, sont les meilleurs résultats obtenus pour ce problème bicritère.

Remarque 4.1. Le problème d'optimisation bicritère a été largement étudié car des résultats très satisfaisants ont pu être obtenus grâce, notamment, à l'avantage de disposer d'un espace de recherche à deux dimensions. L'ajout d'une contrainte dans le plan divise simplement l'espace de recherche en deux sous-espaces. En revanche, dès que le nombre de critères augmente, le nombre de zones à explorer augmente de manière exponentielle.

C'est pourquoi les nombreux résultats proposés pour la recherche des solutions efficaces en optimisation bicritère ne s'étendent généralement pas (ou alors au détriment de l'efficacité des méthodes) à plus de deux critères (voir par exemple Sourd et al. [62]).

Chapitre 5

Méthode exacte pour la résolution du problème de l'arbre multiobjectif

Compte tenu de l'absence de méthodes exactes pour la recherche de l'ensemble des arbres efficaces et/ou non dominés pour le problème de l'arbre multiobjectif ($MOST$) avec plus de deux critères, nous nous sommes attelés à la mise en œuvre d'une méthode exacte pour le problème $MOST$ considérant au moins deux critères .

5.1 Une méthode exacte de résolution de MOST

Dans cette étude, une méthode exacte est présentée pour le problème de l'arbre multiobjectif. Elle est basée sur un principe de séparation par rapport à des arêtes particulières du graphe, induisant une étape de construction des contraintes du problème, de proche en proche, dans une structure arborescente. Ceci a pour effet de partitionner le graphe initial en sous graphes, chacun correspondant à un programme multiobjectif linéaire discret permettant de trouver des arbres efficaces du problème [6, 7].

On rappelle que Le modèle mathématique associé au problème $MOST$ est donné comme suit :

$$x_i = \begin{cases} 1 \text{ si l'arête } e_i \in T; \quad i = \overline{1,m} \\ 0 \text{ sinon} \end{cases}$$

$$(P) \begin{cases} MinZ_1 = \sum_{i=1}^{m} c_{i1} x_i \\ MinZ_2 = \sum_{i=1}^{m} c_{i2} x_i \\ \quad \vdots \\ MinZ_r = \sum_{i=1}^{m} c_{ir} x_i \\ \sum_{i=1}^{m} x_i = n - 1 \quad ..... \quad (1) \\ \sum_{e_i \in E(S)} x_i \leq |S| - 1, \forall S \subset V, S \neq \emptyset \quad ... \quad (2) \\ x_i \in \{0,1\} \quad \forall i = \overline{1,m} \end{cases}$$

5.1.1 Principe de la méthode exacte

Nous commençons par rappeler une propriété importante dans les graphes, notamment une arête d'un graphe $G = (V, E)$ est soit un isthme (une arête dont la suppression augmente le nombre de composantes connexes de G), soit elle appartient à un cycle de G (voir théorème 3.1). Notre méthode est basée sur cette propriété en ce sens que l'obtention de tous les arbres de G se fait en disséquant les arêtes des cycles de G, en particulier ceux de type $e2c$. En effet, l'appartenance ou non de ces arêtes à un arbre T empêche la création de cycles dans T. L'approche proposée est de type séparation et construction dans une arborescence de recherche structurée et construite en utilisant un parcours en profondeur. La séparation se fait sur les arêtes $e2c$ de G et induit l'étape de construction pour décrire les contraintes (2) du programme (P) qui assure l'élimination des cycles par rapport aux arêtes $e2c$. Chaque branche de l'arbre de recherche est construite en agissant alternativement sur les étapes de séparation et construction, qui se termine en une feuille f sans aucune arête de type $e2c$.

Donc, à chacune de ces feuilles f, on obtient un sous-graphe H avec deux possibilités. Dans le premier, les arêtes de H constituent un arbre réduit à la solution unique de la branche correspondante. La deuxième possibilité est que H ne contienne que des cycles arêtes-

disjoints, auquel cas les contraintes éliminant tous ces cycles sont ajoutées au système de contraintes précédemment établi. L'ensemble de contraintes obtenu avec les variables binaires et les critères du programme (P), constitue un programme linéaire multiobjectif $(MOLBP)$ noté (P_f) qui peut être résolu en utilisant l'une des méthodes existantes dans la littérature pour générer l'ensemble $SMST_f$ des arbres efficaces et / ou de l'ensemble SND_f de vecteurs de coûts non dominés associés en la feuille f. Pour fournir une étude expérimentale, on envisage adapter la plus rapide des deux méthodes décrites dans [1] et [51] pour générer l'ensemble SND des solutions non dominées du problème $MOST$. Nous rappelons que le but de ce travail n'est pas de résoudre un programme $MOLBP$ mais de décrire une approche générale pour le problème $MOST$ appliquant simultanément les propriétés de la théorie des graphes et de la programmation linéaire.

Pour rendre rapide notre algorithme basé sur la méthode "Branch and Bound", un ensemble SUB bornant supérieurement, est utilisé. Cet ensemble contient les solutions non dominées trouvées en optimisant les sommes pondérées des critères (solutions efficaces extrêmes [66]) et des solutions potentiellement non-dominés, c'est-à-dire qui ne sont pas dominées par déjà connue, générées par un algorithme de VNS [42]. Fondamentalement, l'algorithme VNS est basé sur le changement systématique du voisinage des solutions générées pour éviter les minimums locaux et s'échapper des vallées qui les contiennent. Pour adapter cette idée au problème $MOST$, nous considérons une population de solutions, où la première population est générée aléatoirement à l'aide de l'algorithme de Kruskal. VNS est appliquée à chacune des solutions actuelles potentiellement non dominées de l'ensemble SUB. À chaque étape de l'algorithme, l'ensemble bornant SUB est tenu à jour au moyen d'une comparaison paire à paire des solutions de SUB et SND_f. Ainsi, l'ensemble SUB est composé de solutions potentiellement non dominées. À la fin de l'algorithme, l'ensemble SUB doit donc correspondre à SND.

De plus, il est facile de calculer l'arbre minimal mono-objectif permettant d'obtenir les coordonnées de points idéaux, utilisées comme limite inférieure dans l'algorithme pour sonder les nœuds de l'arbre de recherche. Si un point de SUB domine ce point idéal, alors le nœud est sondé par dominance. Si ce n'est pas le cas, alors les nœuds fils, dans lesquels une variable binaire plus des contraintes associées sont fixées, sont créés pour que les sous-problèmes correspondants soient explorés.

5.2 Procédures de l'algorithme

Dans cette section, les étapes de l'algorithme $MOST$ sont présentées en détail [6].

1. Procédure SUB

Entrées $G = (V, E)$: graphe connexe, coûts des arêtes de G.

Sorties L'ensemble initial SUB des solutions potentiellement non dominées

Étape 1.1. Générer Ψ_1 un ensemble d'arbres supportés en utilisant une fonction scalarisante.

Étape 1.2. Générer Ψ_2 un ensemble de solutions potentiellement non dominées en utilisant l'algorithme VNS adapté. Pour ce faire, nous ne considérons que deux types de structures de voisinage telles que les voisins d'un arbre T sont obtenus en changeant k arêtes de T qui n'appartiennent pas à un même cycle de G, $k = 1$ ou $k = 2$.

Étape 1.3. Ensemble initial SUB est composé des solutions non dominées de l'ensemble $\Psi_1 \cup \Psi_2$.

2. Procédure Réduction

Entrées $G = (V, E)$: graphe connexe avec $|V| = n$ et $|E| = m$, coûts des arêtes de G.

Sorties G' : un graphe partiel du graphe G, B : base cycle de G', F : un ensemble des arête $e2c$ de G'.

Étape 2.1. Trouver une base de cycle $B = \{\mu_1, \mu 2, .., \mu_{m-n+1}\}$.

Étape 2.1.1. S'il existe une arête e appartenant à un cycle μ de la base B telle que le vecteur-coût de e soit dominé par tous ceux des arêtes de μ, supprimer e du graphe G pour obtenir un graphe partiel G' (voir proposition 5.1).

Étape 2.1.2. S'il existe une arête e appartenant à un ensemble Δ de cycles de B telle que le vecteur- coût de e domine tous ceux des arêtes de Δ, alors l'arête e appartient à tous les arbres non dominés du problème $MOST$ (voir proposition 5.2).

Étape 2.2. Déterminer une nouvelle base B.

Étape 2.3. Trouvez l'ensemble $F = \{e_0 \in E / \exists \mu_l \ \ and \ \ \mu_{l'} \in B; e_0 \in \mu_l \cap \mu_{l'}\}$.

3. Procédure Séparation et Construction

Entrées G', B, F et SUB. (sans perte de généralité, on suppose que G' est d'ordre n et de taille m)

Sorties Un ensemble de contraintes linéaires modélisant l'ensemble des arbres de la branche qui se termine à la feuille courante f.

Étape 3.1.

a- Si $B = \{\mu_1, \mu_2, .., \mu_{m-n+1}\}$ et $F = \emptyset$, on est donc sur feuille f, donc ajouter l'ensemble des contraintes pour casser tous les cycles arêtes-disjointes dans B :

$$\begin{cases} \sum_{e_j \in \mu_i} x_j \leq |\mu_i| - 1, \forall i = \overline{1, m-n+1} & (3) \\ \sum_{e_i \in E} x_i = n - 1 & (4) \end{cases}$$

b- Si $B = \emptyset$, $F = \emptyset$ et $(\sum_{e_i \in E} x_i = n - 1)$, nous sommes en la présence d'une feuille f, on obtient alors un arbre. Sonder cette feuille.

c- Si $F \neq \emptyset$ alors, aller à l'étape 3.2.

Étape 3.2. Sélectionner une arête $e_i \in F$ et créer deux nœuds.

Étape 3.2.1. Dans le premier nœud, ajouter la contrainte suivante à l'ensemble des contraintes précédemment imposées au nœud parent : $x_i = 1$ (5)

a- Si l'arête e_i crée un cycle avec l'ensemble des arêtes déjà fixées, alors ce nœud est sondé.

b- Sinon, ajouter les contraintes suivantes pour détruire les cycles μ_l et $\mu_{l'}$ contenant l'arête e_i :

$$\begin{cases} \sum_{e_j \in \mu_l} x_j \leq |\mu_l| - 1 & (6) \\ \sum_{e_j \in \mu_{l'}} x_j \leq |\mu_{l'}| - 1 & (7) \\ x_j \in \{0, 1\} \quad j = \overline{1, |\mu_l| + |\mu_{l'}| - 1} \end{cases}$$

Étape 3.2.2. Dans le second nœud, ajouter la contrainte suivante à l'ensemble des contraintes précédemment imposées au nœud parent : $x_i = 0$ (8)

Considérer le graphe $G' = G \setminus \{e_i\}$ et aller à **Procédure Réduction**.

Aller à l'étape 3.1.

L'étape suivante correspond à une feuille f dans laquelle on récupère un programme linéaire

multiobjectif avec des variables binaires (P_f) écrit sous la forme générale donnée ci-dessous.

4. Procédure Évaluation

Entées Un programme linéaire en variables binaires à objectifs multiples.

Sorties SND_f l'ensemble des vecteurs-coûts non dominés et / ou $SMST_f$ l'ensemble des arbres efficaces associés.

Étape 4.1 Résoudre le programme (P_f) suivant. Cela se fait en utilisant la méthode [51] pour les programmes dont le nombre de critères est inférieur ou égal à quatre. Pour un plus grand nombre de critères, la méthode citée dans [1] est utilisée.

$$(P_f)\begin{cases} MinZ_i = \sum_{j=1}^{m} c_{ji}x_j \quad ; i = \overline{1,r} \\ \sum_{e_j \in \mu_l} x_j \leq |\mu_l| - 1; l \in L \\ \sum_{e_j \in E} x_j = n - 1 \\ x_p = 1; p \in M \\ x_q = 0; q \in N \end{cases}$$

où le long de la branche de la feuille f, M désigne l'ensemble des indices des arêtes appartenant à tous les arbres efficaces, N désigne l'ensemble des indices des arêtes exclues de tous les arbres efficaces et L l'ensemble des indices des cycles de la base construite.

Étape 4.2. Retourne l'ensemble SND_f et / ou $SMST_f$.

Dans ce qui suit, nous présentons notre algorithme « Algorithme-$MOST$ » pour résoudre le problème $MOST$. Cet algorithme principal appelle toutes les procédures présentées précédemment pour obtenir l'ensemble de tous les arbres non-dominés.

5. Algorithme-$MOST$

Entrées $G = (V, E)$: graphe connexe, C : vecteur-coût des arêtes de G.

Sorties SND : liste des solutions non dominées du problème $MOST$.

Étape 5.1. Exécuter **Procédure SUB**

Étape 5.2. Exécuter **Procédure de réduction**

Étape 5.3. Tant qu'il existe encore des nœuds non sondés dans l'arborescence de recherche, faire :

Étape 5.3.1. Choisir un nœud en fonction de la stratégie en profondeur d'abord.

Étape 5.3.2 Déterminer les coordonnées du point idéal I correspondant à ce nœud.

Étape 5.3.3. Comparer I avec les éléments de l'ensemble SUB :

Étape 5.3.3.1. Si I est dominé par un élément z de l'ensemble SUB, alors stériliser ce nœud et aller à l'étape 5.3.

Étape 5.3.3.2. Sinon, aller à l'étape 5.3.4.

Étape 5.3.4. Exécuter **Procédure séparation et construction**

Étape 5.3.4.1. Si on obtient à la feuille f un arbre T_f, alors : $SND_f := C(T_f)$. Stériliser ce nœud et aller à l'étape 5.3.

Étape 5.3.4.2. Procédure Évaluation

Pour tous les vecteurs-coûts C_T dans SND_f, faire :

a- Si C_T n'est pas dominée par z, $\forall z \in SUB$, alors $SUB := SUB \cup \{C_T\}$.

b- Si $\exists z \in SUB$ dominé par C_T, alors $SUB := SUB \setminus \{z\} \cup \{C_T\}$. Stériliser ce nœud et aller à l'étape 5.3.

Étape 5.4. Terminer, $SND := SUB$.

5.3 Principaux résultats

Dans cette section, les résultats suivants sont prouvés pour justifier la convergence de l'algorithme-$MOST$.

Soit $G = (V, E)$ un graphe connexe et non-orienté d'ordre n et de taille m. Les propositions suivantes permettent de mettre en évidence des arêtes particulières du graphe G dont l'appartenance ou non à des arbres non dominés est prouvée selon la Procédure de Réduction de l'algorithme-$MOST$.

Proposition 5.1. *Soit μ un cycle de G et e une arête de G, $e \in \mu$. Si tous les vecteurs-coûts des autres arêtes du cycle μ dominent celui de l'arête e, alors e ne peut appartenir à aucun arbre non-dominé. [6]*

Preuve 2. *Soit T un arbre et $e \in \mu$, μ un cycle de G tels que tous les vecteurs-coûts des autres arêtes du cycle μ dominent celui de l'arête e et supposons que $e \in T$. Soit $f \in \mu$ et*

$f \notin T$, alors $T' = T \cup \{f\} \setminus \{e\}$ est un arbre puisque ; $e \in T \cup \mu$ et $f \in \mu$ mais $f \notin T$. Alors, $C_k(T') = C_k(T) + c_{fk} - c_{ek}, \quad \forall k = \overline{1,r}$. Par conséquent, $C_k(T') \leq C_k(T), \quad \forall k = \overline{1,r}$, avec au moins une inégalité stricte car le vecteur-coût de f domine celui de e. Alors le vecteur-coût de T' domine celui de T.

Proposition 5.2. *Soit e une arête de G commune à un ensemble Δ de cycles de G et supposons que le vecteur-coût de e domine tous les autres vecteurs-coûts des arêtes des cycles de Δ, alors l'arête e appartient à tous les arbres non dominés de G. [6]*

Preuve 3. *On note $E(\Delta)$ l'ensemble de toutes les arêtes dans Δ. S'il existe un arbre non-dominé T qui ne contient pas l'arête e, alors $T' = T \cup \{e\} \setminus \{f\}, \forall f \in E(\Delta)$ est un arbre dont le vecteur-coût domine celui de l'arbre T parce que l'arête e domine l'arête $f, \forall f \in E(\Delta)$.*

Remarque 5.1. Soit B une base de cycles d'un graphe donné, la façon de briser les cycles de B dépend de l'existence ou non des arêtes de type $e2c$ afin d'associer les contraintes linéaires spécifiques dans la Procédure de séparation et de construction. Pour justifier l'inexistence des arêtes $e2c$, nous prouvons le théorème suivant.

Théorème 5.3. *Si G admet une base de cycles B telle que tous les cycles sont arêtes-disjoints, alors il n'existe pas une autre base de cycles B' dont au moins deux cycles ont des arêtes communes. [6]*

Preuve 4. *Supposons qu'il existe une base de cycles B' telle qu'il existe deux cycles μ_1 et μ_2 dans B', avec μ_1 et μ_2 ayant au moins une arête commune et $\mu_1 \neq \mu_2$.*
Soit $B = \{\sigma_i, i = \overline{1, m-n+1}\}$ une autre base de cycles G.
Comme $\mu_1 \cap \mu_2 \neq \emptyset$, alors $\exists e_j \in E$ tel que $e_j \in \mu_1 \cap \mu_2$.
D'autre part, dans la base B les vecteurs représentatifs des cycles μ_1 et μ_2 sont écrits d'une manière unique selon ceux des cycles de B :

$$\begin{cases} V(\mu_1) = \sum_{i=1}^{m-n+1} \alpha_i V(\sigma_i) \\ V(\mu_2) = \sum_{i=1}^{m-n+1} \beta_i V(\sigma_i) \end{cases}$$

tel que : $V_j(\mu_k) = 1$ if $e_j \in \mu_k$, 0 sinon, $k = 1, 2$ et $\quad \alpha_i, \beta_i \in \{-1, 0, 1\}$.
Comme les cycles $\sigma_i, i = \overline{1, m-n+1}$ sont élémentaires arêtes-disjoints, il n'y a qu'un

cycle σ_{i_0} Contenant l'arête e_j ce qui prouve que $\sigma_{i_0} = \mu_1$ et $\sigma_{i_0} = \mu_2$. Contradiction avec $\mu_1 \neq \mu_2$.

Le théorème suivant prouve la convergence de l'algorithme-$MOST$.

Théorème 5.4. *L'algorithme-MOST est capable de construire tous les arbres non dominés en un nombre fini d'itérations. [6]*

Preuve 5. *Il est évident que l'étape de séparation de la procédure séparation et construction assure l'indépendance des problèmes obtenus aux nœuds de l'arbre de recherche et donc la non-redondance des arbres trouvés, tandis que l'étape de construction met en évidence les contraintes qui assurent que les arêtes d'un cycle donné μ de G ne devraient pas toutes être prises ensemble, sinon les solutions obtenues ne seraient pas des arbres.*
Ceci conduit à trouver toutes les contraintes linéaires du problème $MOST$ à chaque feuille f de l'arborescence de recherche, ce qui permet à la Procédure d'évaluation de déterminer l'ensemble non-dominé SND_f, puis l'ensemble des arbres potentiellement non dominés SUB est mis à jour par conséquent.
Le nombre de feuilles créées étant fini, donc, l'ensemble de tous les arbres non dominés SND est obtenu en un nombre fini d'itérations.

Exemple didactique Considérons le graphe $G = (V, E)$ où $V = \{1, 2, 3, 4, 5, 6\}$ et $E = \{12, 13, 15, 23, 34, 36, 45, 56\}$. Trois coûts sont associés à chaque arête comme indiqué dans la matrice suivante des coûts :

$$C = \begin{pmatrix} 0 & 2 & 1 & 2 & 4 & 0 & 1 & 3 \\ 3 & 3 & 2 & 2 & 3 & 2 & 1 & 1 \\ 1 & 0 & 3 & 2 & 4 & 0 & 1 & 2 \end{pmatrix}$$

Le graphe G est donné dans la figure ci-dessous :

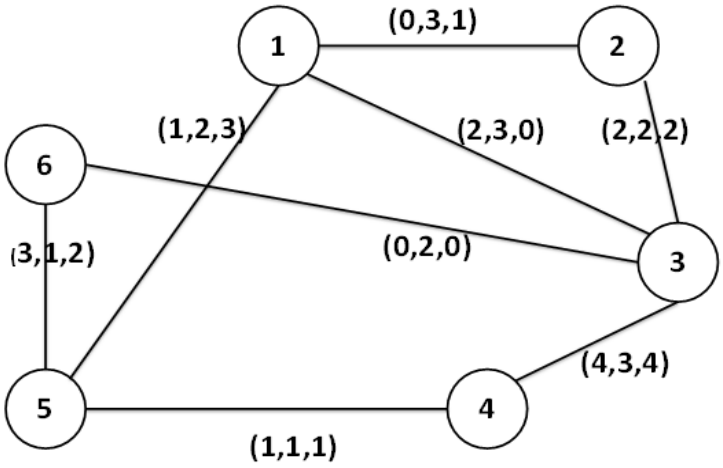


FIGURE 5.1 – Un graphe connexe G

Procédure SUB.

$ST_1 = \{12, 13, 15, 45, 36\}, Z(ST_1) = (4, 11, 5)$.

$ST_2 = \{12, 13, 45, 65, 36\}, Z(ST_2) = (6, 10, 4)$,

$ST_3 = \{12, 15, 45, 65, 36\}, Z(ST_3) = (5, 9, 7)$,

$ST_4 = \{15, 23, 45, 56, 36\}, Z(ST_4) = (7, 8, 8)$.

$SUB = \{Z(ST_1), Z(ST_2), Z(ST_3), Z(ST_4)\}$.

Procédure Réduction

Soit la base de cycles $B = \{\mu_1, \mu_2, \mu_3\}$ avec $\mu_1 = \{13, 36, 65, 51\}$ et $\mu_3 = \{43, 36, 65, 54\}$.

- Dans le cycle μ_3 le vecteur-coût de l'arête 34 est dominé par ceux de toutes les arêtes de μ_3. Selon la proposition 5.1, cette arête est supprimée du graphe G et $E := E \setminus \{34\}$.

- Déterminer alors une nouvelle base de cycles et trouver l'ensemble :

$$F = \{ij \in E / \exists\ \mu_l \ et\ \mu_{l'} \in B;\ ij \in \mu_l \cap \mu_{l'}\}.$$

Soit $B = \{\mu_1, \mu_2\}$ avec $F = \{13\}$.

Procédure Séparation et Construction

$$x_{ij} = \begin{cases} 1 \ \text{ si l'arête } \ ij \in T, \quad i, j = \overline{1, n} \\ 0 \ \text{ sinon} \end{cases}$$

La séparation se fait par rapport à l'arête 13 en créant deux nœuds. Au nœud 1, on déduit le système suivant :

$$(S_1)\begin{cases} x_{13} = 1 \\ x_{36} + x_{65} + x_{15} \leq 2 \\ x_{32} + x_{21} \leq 1 \\ x_{12} + x_{15} + x_{23} + x_{36} + x_{45} + x_{65} = 4 \\ x_{12}, x_{13}, x_{15}, x_{23}, x_{36}, x_{45}, x_{65} \in \{0, 1\} \end{cases}$$

Procédure Évaluation (au nœud 1)

$F := \emptyset$, Alors le noeud 1 est une feuille. En utilisant la méthode décrite dans [51] pour la résolution d'un programme linéaire multiobjectif en des variables binaires (P_1) dont le système de contraintes est (S_1), Il renvoie tous les arbres efficaces associés à cette branche de l'arborescence de recherche :

$T_1 = \{12, 13, 45, 65, 36\}, Z(T_1) = (6, 10, 4)$,

$T_2 = \{13, 23, 45, 65, 36\}, Z(T_2) = (8, 9, 5)$,

$T_3 = \{12, 13, 15, 45, 36\}, Z(T_3) = (4, 11, 5)$.

$SND_{f_1} = \{Z(T_1), Z(T_2), Z(T_3)\}$.

$SUB = SUB \cup Z(T_2)$.

Procédure Réduction

Au niveau du nœud 2 de l'arbre de recherche, l'arête 13 est supprimé et $E := E \setminus \{13\}$. Le point idéal I correspondant à ce nœud est donné par : $Z(I) = (4, 8, 6)$ qui n'est pas dominé par les solutions trouvées précédemment dans le premier nœud, d'où, ce deuxième nœud n'est pas stérilisé.

La nouvelle base $B = \mu_4$ est obtenue avec $\mu_4 = \{12, 23, 36, 65, 51\}$.

Procédure Séparation et Construction : Le système de contraintes linéaires correspondant est :

$$(S_2)\begin{cases} x_{13} = 0 \\ x_{12} + x_{23} + x_{36} + x_{65} + x_{51} \leq 4 \\ x_{12} + x_{15} + x_{23} + x_{36} + x_{45} + x_{65} = 5 \\ x_{12}, x_{13}, x_{15}, x_{23}, x_{36}, x_{45}, x_{65} \in \{0, 1\} \end{cases}$$

Procédure Évaluation (au nœud 2)

Dans cette feuille, la méthode décrite dans [51] est utilisée pour résoudre le programme linéaire multiobjectif en variables binaires (P_2) associé à (S_2). Tous les arbres efficaces

associés à cette branche sont : $T_4 = \{12, 23, 45, 65, 36\}, Z(T_4) = (6, 9, 6)$,

$T_5 = \{12, 15, 45, 65, 36\}, Z(T_5) = (5, 9, 7)$,

$T_6 = \{12, 23, 15, 45, 36\}, Z(T_6) = (4, 10, 7)$,

$T_7 = \{15, 23, 45, 56, 36\}, Z(T_7) = (7, 8, 8)$.

$SND_{f_2} = \{Z(T_4), Z(T_5), Z(T_6), Z(T_7)\}$.

$SUB = SUB \cup \{Z(T_4), Z(T_6)\}$.

Ainsi, l'ensemble efficace final des arbres de G est : $SMST = \{ST_1, ST_2, ST_3, ST_4, T_2, T_4, T_6\}$ et l'ensemble des solutions non dominées associées est : $SND = SUB$.

Exemple 5.1. On présente aussi un exemple donné dans l'article [2] dont le graphe est celui présenté dans la figure ci-dessus : Ce graphe a 7 sommets et 13 arêtes avec $r = 2$,

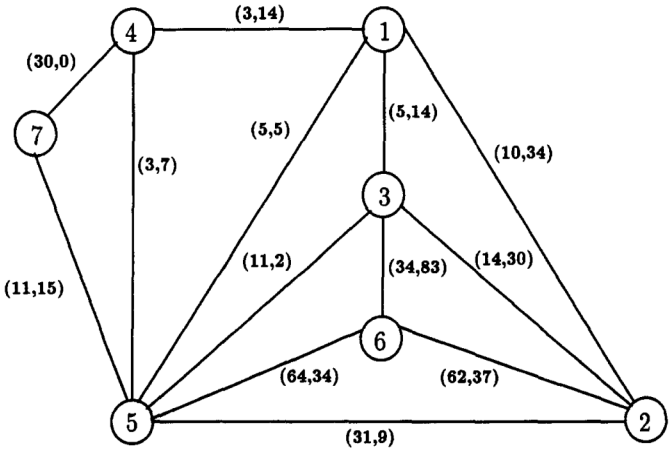


FIGURE 5.2 – Un graphe connexe G traité dans [2]

est connexe et contient 7 cycles élémentaires. Dans [2], les auteurs ont mentionné que ce graphe contient 480 arbres dont 7 sont des arbres efficaces supportés et 15 efficaces non supportés.

On a déroulé notre algorithme $MOST$-Algorithme pour ce graphe, l'ensemble des 22 arbres efficaces a été obtenu en un temps négligeable (tend vers 0 seconde).

5.4 Résultats expérimentaux

L'étude expérimentale est réalisée sous Matlab 2014a sur un ordinateur portable hp, Intel Core i5, 8 Go de RAM, sur des instances générées aléatoirement. Dans ce cas, toutes les instances générées considèrent les vecteurs de coûts de dimensions 3, 4, 5, qui sont

uniformément répartis dans l'intervalle $[-10, 50]$. Les graphes sont générés de façon aléatoire comme décrit par Erdös et al. dans [29] proposent à l'utilisateur de fixer le nombre de sommets du graphe et la probabilité qu'une arête existe entre deux sommets. Pour ce faire, nous avons jugé utile de grouper dans chaque ligne du tableau 1 les résultats moyens pour vingt (20) graphes pour lesquels le nombre d'arêtes varie dans un même intervalle de longueur ne dépassant pas dix (10).

r	n	m	$nbSND$			$CPU(s)$			Cub	nbL
			avg	min	max	avg	min	max	avg	avg
3	20	[35,45]	38,1	18	102	30,7	10,4	87,9	0,04	301,2
3	20	[46,55]	67	35	105	115,9	45,2	216,8	0,1	595,4
3	30	[45,55]	121,3	51	186	337,1	83	858,2	0,08	2240,8
3	30	[56,65]	273,2	87	403	1203	703,2	2301,7	0,5	2603
3	50	[65,75]	881,6	107	1025	2489,9	1833,2	3581,7	1,2	4843,2
4	20	[35,45]	102	66	222	44,2	28,2	178	1,17	356,4
4	20	[46,55]	104,3	44	130	186,7	70,3	227,7	1,3	660
4	30	[45,55]	188,5	63	305	557,4	112,2	1038,4	2,6	2452,6
4	30	[56,65]	320,3	94	690	1723,9	921,1	3074	3,02	3008,7
4	50	[65,75]	1060,2	230	2003	3030,68	2100,4	3864	4,6	5036,4
5	20	[35,45]	246	87	440	88,2	47,2	206	4,8	489,5
5	20	[46,55]	274,8	105	473	294,5	161,6	924,9	5,01	1090,8
5	30	[46,55]	758,3	123	1053	1033,4	452,6	2785	5,21	2821,3
5	30	[55,65]	930,3	202	2066	2034,7	1132,7	3520,3	7,09	3140,6
5	50	[65,75]	1300,4	314	2135	3423,1	2306,1	4037,8	7,9	5453,5

TABLE 5.1 – Résultats expérimentaux

Dans le tableau 5.1, nous avons enregistré la moyenne, le minimum et le nombre maximal de solutions non dominées $nbSND$, ainsi que le CPU en secondes, le Cub qui représente le temps d'exécution utilisé pour générer l'ensemble initial SUB et le nombre moyen de feuilles (nbL).

Pour les cas avec trois et quatre critères, les meilleurs résultats sont obtenus en utilisant

la méthode décrite dans [51], alors que tous les autres résultats sont obtenus en utilisant la méthode décrite dans [1] qui est plus rapide avec un nombre croissant de critères.

Les résultats montrent clairement l'augmentation du temps CPU par rapport au nombre d'arêtes et celle des feuilles (nbL). Cependant, le nombre de critères ne semble pas beaucoup agir sur ce temps et l'évaluation exacte du point idéal en chaque nœud de l'arborescence de recherche a apporté une amélioration à la méthode puisque le nombre de nœuds stérilisés représente une moyenne de 55.63% du nombre total de feuilles créées (nbL).

De plus, l'ensemble SUB considéré comme borne supérieure pour l'ensemble SND fixé le long des étapes de l'algorithme-$MOST$, évolue de manière dynamique et est mis à jour dans un temps polynomial. En effet, l'ensemble SND contient des solutions potentiellement non dominées que nous mettons à jour pendant la recherche. En particulier, tout élément de cet ensemble pourrait être supprimé s'il est dominé par une nouvelle solution récemment rencontrée.

Algorithme qui qui génère des graphe avec un petit nombre d'arête de type $e2c$

Entrée : n : le nombre de sommets du graphe G, $s = 0, p = 0, j = 1$.

Sortie : E : l'ensemble des arêtes de G.

Étape 1. : Générer aléatoirement un nombre $i \in [3, \frac{n}{4}]$

Étape 2. : Tant que $s < n$ faire :

Étape 2.1. : Générer un cycle élémentaire μ_1 qui passe par les sommets $v_j, v_{j+1}, .., v_i$

Étape 2.2. : poser $j = i$

Étape 3. : Générer aléatoirement un nombre $i \in [j, n]$.

Étape 3.1. : Générer un cycle élémentaire μ_k qui passe par les sommets $v_j, v_{j+1}, .., v_i$, et on note $E\mu_k$ l'ensemble des arêtes du cycle μ_k.

Étape 4. : s=le plus grand indice des sommets des cycles déjà construits.

Étape 5. : $p = p + 1$.

Étape 5.1. : $E = E \cup \{E(\mu_k)\}$.

Fin tant que

Étape 6. : Générer aléatoirement un ensemble d'arêtes F de type $e2c$, tel que $|F| \leq 7$.

Étape 6.1. : $E = E \cup \{F\}$.

Un algorithme ci-dessus est utilisé pour la génération des graphes du tableau 5.2 :

r	n	m	$nb\ e2c$	$CPU(s)$	$nb\ SND$
3	100	130	3	47,3	148
3	300	340	4	956,24	566
3	400	445	5	1030,32	759
4	100	130	3	1404,38	768
4	300	340	3	1521,12	1095
4	400	445	5	2234,81	1155

TABLE 5.2 – Résultats en moyenne pour les graphes ayant un petit nombre d'arêtes de type $e2c$

Le temps de calcul de l'algorithme-$MOST$ diminue avec un nombre décroissant $nb\ e2c$ des cycles arêtes-disjoints dans G. En effet, les graphes avec un faible nombre $nb\ e2c$ ont également été considérés. Dans ce cas, les résultats rapportés dans le tableau 5.2 montrent que l'algorithme $MOST$ est plus rapide, ce qui permet de traiter des instances de dimensions plus grandes.

Au cours de notre expérimentation, le calcul est interrompu pour les instances nécessitant plus de 2 heures de calcul.

Chapitre 6

Une méthode approchée GA-VNS pour MOST

Le passage au cadre multiobjectif rend explicitement les problèmes plus ardus, en particulier du fait du nombre très grand de solutions efficaces. Cela est aussi vrai pour le problème de l'arbre minimum. Ainsi, alors que des algorithmes pour le cas mono-objectif peuvent résoudre des instances ayant plusieurs centaines de milliers de variables en un temps raisonnable, les résultats disponibles pour les algorithmes de la version multiobjectif de $MOST$ se cantonnent à deux critères au plus. De plus, la méthode exacte que nous avons proposée dans le chapitre précédent arrive juste à résoudre des instances moyennes de graphes de l'ordre d'une centaine d'arêtes, en des temps raisonnables. Il est alors naturel de s'orienter vers des méthodes approchées pour pouvoir résoudre des instances de grande taille. C'est dans cette optique que plusieurs métaheuristiques ont été appliquées au problème $MOST$. Nous nous limitons à mentionner les principales contributions existantes.

L'algorithme déterministe des points extrêmes (EPDA) a été proposé [21] pour trouver à la fois des solutions efficaces supportées et non supportées pour plus de deux critères. Cet algorithme est validé à l'aide d'un algorithme de recherche exhaustive, basé sur la méthode proposée par Christofides [13], en utilisant des instances de benchmarks générées par des algorithmes suggérés par Knowles [46]. Ces derniers se composent de graphes complets ayant trois fonctions de coûts différentes.

Lorsque le nombre de nœuds est important et que les algorithmes déterministes sont lents

à converger et deviennent impraticables, des algorithmes probabilistes peuvent être utilisés pour trouver de bonnes représentations des fronts Pareto en moins de temps. Les quelques algorithmes évolutifs proposés dans la littérature ont été testés sur de petits cas, pour des problèmes bicritère et la majorité algorithmes trouve seulement les solutions efficaces supportées.

Zhou et Gen [72] semblent avoir été les premiers à suggérer d'appliquer la première version de l'algorithme génétique "Non-dominated Sorting Genetic Algorithm" (NSGA) [64] au problème *MOST* avec seulement deux critères, afin d'énumérer toutes les solutions Pareto non dominées. Cependant, Knowles [46] a souligné que l'algorithme d'énumération proposé n'était pas correct puisque il retourne des solutions Pareto-dominées et omettre certaines solutions efficaces.

Un algorithme appliquant l'algorithme "Greedy Randomized Adaptive Search Procedure" (mc-GRASP) a été suggéré par Arroyo et al. [3]. L'algorithme est basé sur l'optimisation de différentes fonctions d'utilité pondérées. A chaque itération, un vecteur de poids d'agrégation des critères est défini et une solution est obtenue en utilisant une procédure constructive. La solution trouvée est soumise à une recherche locale en essayant d'améliorer la valeur de la fonction d'utilité pondérée. Afin de trouver une variété de solutions efficaces, les auteurs utilisent différents vecteurs de poids, qui sont répartis uniformément sur la frontière de Pareto. Cet algorithme permet d'obtenir à la des solutions efficaces supportées et non supportées.

Plus récemment, d'autres algorithmes génétiques ont été proposés par Han et Wang [41] et par Chen et al. [12] qui ont été testés sur des graphes de petites tailles avec seulement deux critères.

Gue et al. [37] ont présenté un algorithme d'Essaimes Particulaire pour plus de deux critères, mais aucune expérimentation informatique n'a été rapporté.

Un algorithme appelé "A novel fast and scalable Knowledge-based Evolutionary Algorithm" (KEA) est proposé dans [22], dont les principales caractéristiques sont :

- L'application d'approches déterministes pour calculer les points extrêmes du front Pareto. Celles-ci sont utilisées pour produire la population initiale composée d'un ensemble de parents.
- Une recherche évolutionnaire élitiste tente de trouver les points optimaux de Pareto res-

tants en appliquant un opérateur de mutation. La détermination des solutions est basée sur les approches "k-best" bien connue dans les méthodes déterministes.
- Des systèmes de marquage permettant de réduire la réévaluation des solutions ainsi que des points de coupure qui éliminent des régions dominées de l'espace de recherche sont aussi appliqués.
Cet algorithme peut être applicable à plus de deux critères et calcule à la fois les solutions supportées et non supportées.

Sachant que la méthode décrite dans [22] est la plus récente et la plus efficace par rapport à ce qui existe dans la littérature, d'après les auteurs, on a jugé utile de la comparer avec notre approche qui est détaillée dans la section suivante. On rappelle d'abord les deux méthodes connues dans la littérature, "Non-dominated Sorting Genetic Algorithm-II", et "Variables Neighborhood Search".

1- "Non-dominated Sorting Genetic Algorithm-II" (NSGA-II)

L'algorithme génétique NSGA-II proposé par Deb et al. [23], apparaît comme l'un des algorithmes les plus efficaces pour approximer l'ensemble optimal de Pareto avec une excellente variété des solutions. Il est basé sur les trois caractéristiques suivantes : il utilise le principe de l'élitisme, favorise les solutions non dominées et utilise une variété explicite des solutions.
L'algorithme NSGA-II commence par une génération aléatoire d'une population initiale P_0 de N individus (parents). A la génération t, une population Q_t de N enfants est créée à partir de la population parent P_t en utilisant les opérateurs génétiques (sélection-croisement-mutation). Ensuite, les deux populations sont combinées pour former une nouvelle population R_t de taille $2N$. La recherche des solutions non dominées permet de classer les individus de R_t en plusieurs fronts de rangs différents. Elle est effectuée de la manière suivante : chaque individu de la population R_t est comparé à tous les autres par le concept de dominance. Les individus non dominés appartiennent au front de rang 1, le front de Pareto. En éliminant temporairement ces individus de l'ensemble de recherche, l'algorithme est itéré pour fournir le front de rang 2, et ainsi de suite. La nouvelle population parent P_{t+1} est alors construite avec les N individus appartenant aux fronts de rangs les plus faibles.

Généralement, pour le dernier front il y a plus de solutions que de places restantes dans la nouvelle population P_{t+1}. Les individus sont alors triés selon une distance connue sous le vocable anglais de "crowding distance". Ce choix permet d'offrir la meilleure distribution des individus sur le front le plus élevé.

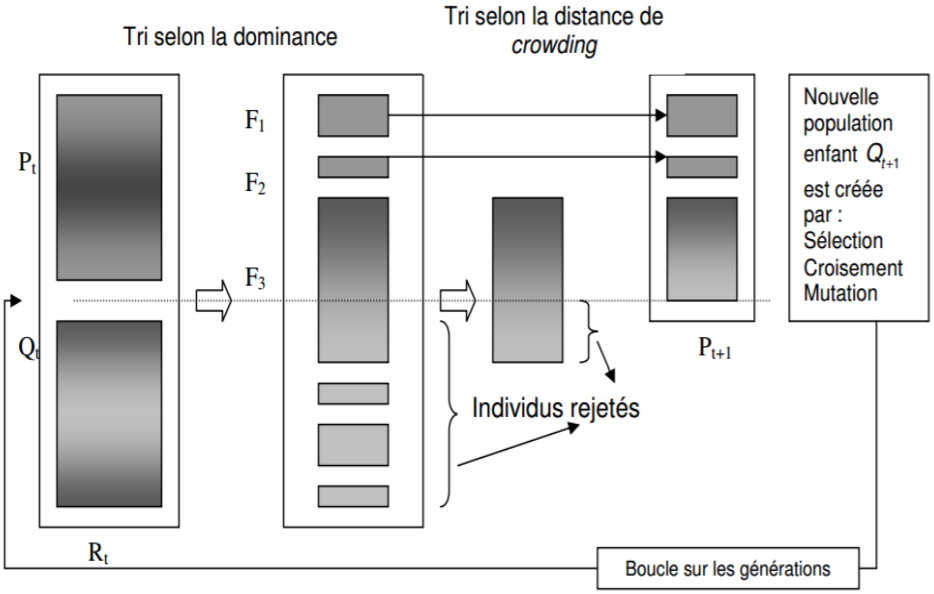


FIGURE 6.1 – Principe de l'algorithme NSGA-II

Calcul de la distance de "crowding"

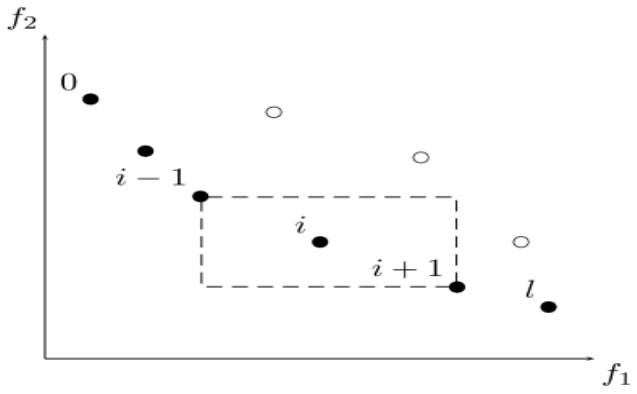


FIGURE 6.2 – Distance de Crowding

La distance de "crowding" d_i d'un point particulier i se calcule en fonction du périmètre de l'hypercube ayant comme sommets les points les plus proches de i sur chaque fonction objectif.

Sur la figure ci-dessus est représenté l'hypercube en deux dimensions associé au point i. Un algorithme de calcul de la distance de "crowding" est détaillé dans [Deb, 2001]. Cet algorithme est de complexité $O(MNlog(N))$, où M est le nombre d'objectifs du problème et N le nombre d'individus à traiter. Une fois tous les d_i calculés, il ne reste plus qu'à les trier par ordre décroissant et à sélectionner les individus possédant la plus grande valeur de "crowding".

Les techniques de sélection en vue d'un croisement ou d'une mutation peuvent aussi bénéficier de ce type de calcul. En effet, l'opérateur de sélection le plus fidèle à l'esprit de Pareto tombe en défaut dès que les deux individus en compétition ont le même rang de dominance (i.e. sont non dominés). Pour pallier à ce problème, NSGA-II utilise la distance de "crowding" pour départager les deux individus. Ce processus permet de conserver les bonnes propriétés de la sélection Pareto tout en préservant une certaine diversité sur les individus sélectionnés.

2- "Variable Neighborhood Search" (VNS)

La Recherche à voisinage variable, proposée par Hansen et Mladenović [42] en 1997, est une métaheuristique récente pour la résolution de problèmes d'optimisation, dont l'idée de base est le changement systématique de voisinage au sein d'une recherche locale. Nous présentons dans ce qui suit, les règles de base de VNS.

L'heuristique VNS utilise les constats suivants :

- Un minimum local par rapport à un voisinage n'en est pas nécessairement un par rapport à un autre
- Un minimum global est un minimum local par rapport à tous les voisinages possibles
- Pour de nombreux problèmes, les minimas locaux par rapport à un ou à plusieurs voisinages sont relativement proches les uns des autres.

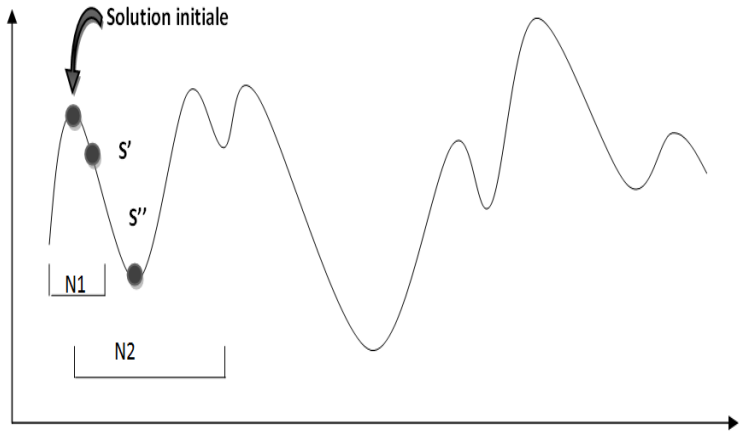


FIGURE 6.3 – Recherche à voisinage variable (Hansen et *al.*, 1997)

6.1 Principe de la méthode approchée GA-VNS

Le but de notre méthode est de trouver une bonne approximation du front de Pareto pour le problème de l'arbre minimum multiobjectif ($MOST$). Nous proposons un algorithme hybride GA-VNS ("Genetic Algorithm-Variable Neighborhood Search") qui combine les avantages des algorithmes NSGA-II et VNS. Cet algorithme commence par la génération d'une population initiale P de solutions réalisables (arbres), adopte un nouvel opérateur de croisement à deux points et applique une heuristique qui décrit l'opérateur de mutation. La population finale obtenue à la fin de NSGA-II après sélection, est améliorée par un algorithme basé sur la méthode VNS. L'expérimentation montre que cette hybridation permet de trouver de bons individus pour contrebalancer entre la diversification et l'intensification au cours du processus de recherche d'optimisation [8, 9].

6.1.1 Adaptation de l'algorithme GA-VNS

1- Codage d'un individu (solution arbre)

Initialement les arêtes du graphe G, d'ordre n et de taille m, sont numérotées. Le codage direct d'un chromosome (correspondant à un arbre) est un ensemble de dimension $n-1$, dans lequel chaque coordonnée représente le numéro associé d'une arête.

Remarque 6.1. Dans l'article [46], les auteurs soulignent que le codage direct est meilleur et plus efficace que le codage de Purfer [35], contrairement à ce que Zhou a montré dans

son article [72].

2- Génération d'une population initiale

La première population de taille p des arbres est générée moyennant les trois procédures suivantes :

- à l'aide des arbres optimaux correspondants à chaque critère,
- par génération aléatoire avec agrégation de critères,
- par l'application de l'algorithme de Kruskal en sélectionnant aléatoirement les arêtes du graphe G.

Remarque 6.2. 1- Nous notons que le processus de construction de la population initiale permet d'obtenir des solutions réalisables et d'adapter un équilibre entre la qualité et diversité des individus.

2- La reproduction consiste à recombiner deux individus par l'opérateur de croisement, éventuellement suivi d'une mutation des gènes des individus. Par conséquent, à partir de la population initiale, une nouvelle population est obtenue. De cette nouvelle population, une seconde nouvelle population est produite par le même processus et ainsi de suite. Le critère d'arrêt est basé sur le nombre de générations.

3- Croisement

Avant de décrire les étapes de croisement des individus, nous présentons une étude comparative entre le croisement à un point et celui à deux points pour notre problème $MOST$.

Remarque 6.3. Comme pour la méthode exacte décrite dans le chapitre précédent, l'étude expérimentale est réalisée sous Matlab 2014a sur un ordinateur portable *hp*, Intel Core i5, 8 Go de RAM, sur des graphes complets d'ordre n et les vecteurs coûts de dimensions r sont générées aléatoirement.

n	r	$CPU1(s)$	$C(C-2p, C-1P)$	$CPU2(s)$	$C(C-1p, C-2P)$
k30	3	18,88	0,50	18,94	0,77
k50	3	63,90	0,60	47,01	0,85
k80	3	96,48	0,57	77,05	0,87
k100	3	220,68	0,48	167,49	0,73
k80	5	281,60	0,19	307,87	0,88
k100	5	321,61	0,33	328,12	0,77

TABLE 6.1 – Comparaison de GA-VNS avec un croisement à un point et celui à deux points

On note $C(C-1p, C-2p)$ la proportion des solutions trouvées par notre algorithme avec un croisement à un point qui dominent celles trouvées par GA-VNS avec un croisement à deux points et $CPU1(s)$ le temps de calcul associé en seconde. $C(C-2p, C-1p)$ la proportion des solutions trouvées par notre algorithme avec un croisement à deux points qui dominent celles trouvées par GA-VNS avec un croisement à un point et $CPU2(s)$ le temps de calcul associé en seconde

Il convient d'observer que 44% des solutions obtenues par GA-VNS avec un croisement à un point dominent les solutions obtenues par GA-VNS avec un croisement à deux points, tandis que 81% des solutions obtenues par GA-VNS avec le croisement à deux points dominent les solutions obtenues par GA-VNS avec un croisement à un point. Ce résultat est très important pour le choix de l'opérateur de croisement . En effet, notre champ d'application est de fournir de nouveaux moyens pour améliorer davantage cet opérateur.

Croisement à deux points :

Il est similaire au croisement à un point, sauf que deux points de coupure sont générés au hasard au lieu d'un seul. Les chromosomes issus obtenus ne correspondent pas forcément à des solutions arbres. Pour surmonter cette infaisabilité, nous avons introduit des procédures de réarrangement des arêtes pour chaque descendant qui n'est pas un arbre.

Le croisement à deux points est décrit comme suit :

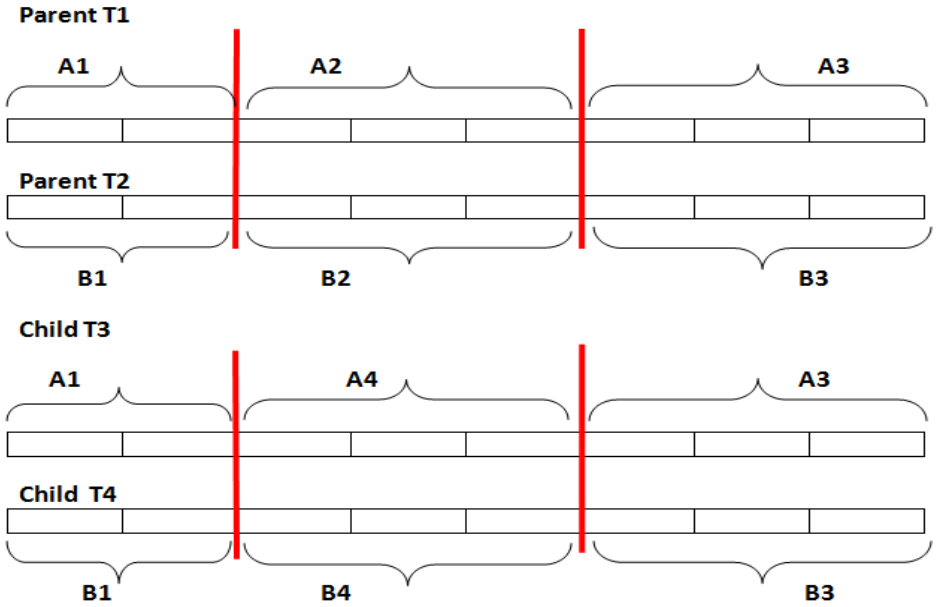


FIGURE 6.4 – Croisement à deux points

Procédure 1 : Choisir les arêtes de $A4$ dans les ensembles, dans l'ordre suivant : $B2$, $B1$, $B3$ et $E \setminus A1 \cup A3$. Ceux de $B4$ des ensembles, dans l'ordre suivant : $A2$; $A1$; $A3$ et $E \setminus B1 \cup B3$.

Procédure 2 Choisir les arêtes de $A4$ dans l'ordre décroissant des sommes de coûts de chaque arête e dans l'ensemble : $E(T2) \cup E \setminus A1 \cup A3$.

Procédure 3 Choisir les arêtes de $A4$ dans l'ordre lexicographique des coûts de chaque arête e dans l'ensemble : $E(T2) \cup E \setminus A1 \cup A3$.

Remarque 6.4. Chaque descendant obtenu par l'opérateur de croisement proposé, est un arbre (solution réalisable), donc une correction ne sera nécessaire pour assurer la faisabilité.

Le tableau ci-dessous présente une comparaison de GA-VNS avec le croisement à deux points en appliquant la procédure 2 et la procédure 3.

n	r	$CPU1(s)$	$C(som, lex)$	$CPU2(s)$	$C(lex, som)$
k80	3	152,13	0,37	186,17	0,46
k100	3	253,08	0,82	255,32	0,57
k200	3	741,33	0,71	776,37	0,55
k80	5	100,59	0,72	147,17	0,53
k100	5	223,01	0,94	268,53	0,46
k200	5	898,46	0,85	945,23	0,33

TABLE 6.2 – Comparaison de GA-VNS avec les procédures 2 et 3

On note $C(som, lex)$ la proportion des solutions trouvées par notre algorithme en appliquant la deuxième procédure, qui dominent celles trouvées par GA-VNS avec en appliquant la première procédure et $CPU1(s)$ le temps de calcul associé en seconde. $C(lex, som)$ la proportion des solutions trouvées par notre algorithme en appliquant la première procédure, qui dominent celles trouvées par GA-VNS avec en appliquant la deuxième procédure, $CPU2(s)$ le temps de calcul associé en seconde

On remarque que la deuxième procédure donne de meilleur résultats (74% par rapport à 48%).

3- Mutation

La mutation est réalisée comme dans le processus biologique évolutionnaire. Son objectif prioritaire est de générer de meilleures solutions, au sens de la domination, que celles de la population actuelle. L'opérateur de mutation est appliquée à chaque descendant dans la population avec une faible probabilité prédéterminée, il consiste à choisir aléatoirement un arête (gène) qui ne se trouve pas dans l'arbre (chromosome) descendant (T) (Une arête $e = (x, y)$ de $G \setminus T$) et le remplacer par une autre arête f dont le vecteur-coût domine celui de e. On obtient un arbre $T1$. L'arête f est dans l'arbre $T1$ et appartient à la chaîne élémentaire C dans $T1$ reliant les sommets x et y, $C \cup e$ est alors un cycle de G. Cela permet de trouver un autre chromosome (c'est-à-dire un arbre) dont le vecteur-coût domine celui de T. S'il n'y a pas d'arête dans la chaîne C dont le vecteur-coût domine celui de l'arête e, on choisit une arête au hasard dans la chaîne C dont le vecteur-coût n'est pas dominé par celui de l'arête e et on remplace cette dernière pour obtenir un nouvel arbre T'.

1. Choisir une arête $e = (x, y)$ de $G \setminus T$.
2. Trouver une chaîne C dans T reliant les sommets x et y ($C \cup e$ est alors un cycle de G).

3. Déterminer une arête f de C dont le vecteur-coût est dominé par celui de e :

– Si f existe, $T' := (T \setminus f) \cup e$ est un arbre qui domine l'arbre fils T.

– Sinon, choisir une arête f de la chaîne C et $T' := (T \setminus f) \cup e$.

5- Sélection

L'opérateur de sélection consiste à procéder d'abord à la partition de la population de taille $2p$, composée de parents et de descendants dans les fronts, le premier contenant les individus non dominés, le second contenant encore des individus non dominés après avoir enlevé les éléments du premier front, et ainsi de suite. La nouvelle population consiste à choisir des individus dans l'ordre de dominance des fronts. Les individus du front k qui doivent compléter la taille p de la nouvelle population sont sélectionnés en fonction d'une distance de "crowding". On note $ASND$ les éléments du premier front.

6- Amélioration des éléments du premier front ASND

Des solutions voisines sont générées pour chaque solution non dominée appartenant à l'ensemble des solutions du premier front de NSGA-II, pour construire une nouvelle population $ASND$, et réaliser la mise à jour de l'ensemble courant non dominé à chaque étape.

La mise à jour du $ASND$ se fait selon la relation de dominance et à chaque itération, les populations obtenues n'ont pas nécessairement la même taille.

Enfin, on maintient une approximation améliorée avec un nombre plus élevé de solutions non dominées. Nous détaillons les étapes de cette amélioration en adaptant l'algorithme VNS comme suit :

- $ASND$: ensemble des solutions non dominées.

- $N_k, k = 1, 2, 3$ est un ensemble de structures de voisinages qu'on appelle aussi $k - opt$, $k = 1, 2, 3$.

- Critère d'arrêt : un nombre fixé d'itérations est atteint,

- $N_k(T)$ est l'ensemble des solutions dans le $k - ième$ voisinage de l'arbre T.

Pour le voisinage $1-opt$ une recherche locale est appliquée, pour le voisinage $k-opt$ chacun des voisins de l'arbre T dans $N_k(T)$ peut être atteint en changeant exactement k arêtes de T.

- Le choix des arêtes, qui ne sont pas des isthmes de G, à changer dans un arbre donné T, se fait de la manière suivante :

- Choisir aléatoirement k arêtes de $G \setminus T$.

- Pour chaque arête $e \in G \setminus T$, trouver la chaîne qui relie des extrémités dans T.

- S'il existe une arête f tel que $C(f) \leq C(e)$, $T = T \setminus \{e\} \cup \{f\}$.

- En utilisant ces trois structures, nous assurons la faisabilité des solutions voisines obtenus(i.e., arbre).

6.2 Résultats expérimentaux

L'objectif de ce travail est de montrer l'efficacité de l'algorithme proposé GA-VNS pour obtenir une bonne approximation du front de Pareto du problème $MOST$. Pour ce faire, l'algorithme GA-VNS est comparé à notre méthode exacte qui génère l'ensemble de toutes les solutions (arbres) non dominées pour des instances de taille moyenne générées aléatoirement.

Nous exécutons GA-VNS et $MOST$-Algorithme pour dix instances de même dimension, et nous comparons les résultats en moyenne en utilisant la mesure de proportion. On note $C(SND, ASND)$: la proportion de solutions de l'ensemble $ASND$ appartenant effectivement à l'ensemble exact des solutions non dominées trouvées par $MOST$-Algorithme.

n	m	r	$SND/ASND$		
			avg	min	max
20	[46,55]	3	67,14%	49,36%	77,81%
30	[56,65]	3	70,89%	50,79%	81%
50	[65,75]	3	68,60%	58,12%	79,06%
20	[46,55]	5	76,27%	56,26%	84,16%
30	[56,65]	5	71,28%	47,70%	88,12%
50	[65,75]	5	68,64%	42,89%	86,24%

TABLE 6.3 – Comparaison entre $MOST$-Algorithme et GA-VNS

D'après ce tableau 6.3 comparatif , on remarque que plus de 70% de solutions non dominées ont été trouvées par notre méthode approchée GA-VNS.

Une autre étude comparative entre notre algorithme et l'algorithme présenté KEA dans [22] est faite avec le même générateur de graphes et les résultats sont présentées dans le tableau

ci-dessous :

n	r	C(GA-VNS,KEA)	CPU1(s)	C(KEA,GA-VNS)	CPU2(s)
K50	3	80%	113,86	36%	206,72
K80	3	78%	204,38	28%	104,21
K100	3	79%	523,31	37%	319,17
K200	3	84%	882,31	38%	842,938
K80	5	85%	296,79	32%	185,395
K100	5	76%	617,93	37%	525,395
K200	5	82%	1160,1	41%	981,13

TABLE 6.4 – Comparaison entre GA-VNS et KEA

On note $C(GA-VNS, KEA)$ la proportion des solutions trouvées par notre algorithme GA-VNS qui dominent celles trouvées par KEA, et $C(KEA, GA-VNS)$ la proportion des solutions trouvées par l'algorithme KEA qui dominent celles trouvées par notre algorithme GA-VNS. $CPU1(s)$ le temps de calcul que dépense GA-VNS et $CPU2(s)$ le temps de calcul de KEA.

Les résultats montrent que 81% de solutions trouvées par notre méthode GA-VNS dominent les solutions trouvées par la méthode KEA tandis que 36% solutions trouvées par notre méthode GA-VNS sont dominées les solutions trouvées par la méthode KEA.

Conclusion et perspectives

Constatant l'absence d'algorithmes exactes pour la résolution de problèmes de l'arbre à objectifs multiples, la motivation centrale de nos travaux fut de proposer des réponses efficaces pour ce problème NP-difficile.

Dans cette étude, on a proposé une méthode exacte basée sur un processus de séparation sur des arêtes qui appartiennent à au moins deux cycles d'un graphe donné. Cela génère une procédure de construction de contraintes qui cassent les cycles tout en conservant la connexité du graphe. Ainsi, en chaque feuille de l'arbre de recherche, le graphe obtenu a une structure particulière ; il ne contient que des cycles arêtes-disjoints, ou bien, il s'agit d'un arbre. Par conséquent, si la première base de cycles du graphe initial ne contient que des cycles arêtes-disjoints, l'algorithme ne crée qu'une seule feuille et un seul programme linéaire bivalent multiobjectif est résolu, ce qui induit un temps d'exécution plus rapide. Sinon à chaque feuille d'une branche arbitraire de l'arborescence de recherche, on résout un programme linéaire bivalent multiobjectif du problème de l'arbre minimum en fonction des contraintes induites le long de cette branche.

A travers ce processus de séparation-construction, nous avons cherché à surmonter les difficultés liées à la résolution directe du programme linéaire bivalent (P) (voir page 54) associé au problème $MOST$. En effet, cette décomposition a permis l'obtention de problèmes de même nature que (P), mais de dimensions beaucoup plus réduites avec, chaque fois, des matrices de contraintes creuses, ce qui conduit à une résolution plus rapide. D'autre part, le calcul des coordonnées du point idéal est un problème facile, on se ramène à la résolution de r problèmes classiques de l'arbre de poids minimum, ce qui a permis de ne pas visiter des zones du domaine des solutions arbres qui ne contiennent pas d'arbres non dominés. Ce résultat apparait clairement à travers l'expérimentation réalisée qui montre,

qu'en moyenne, plus de 50% des nœuds de l'arbre de recherche sont stérilisés avant terme, et cela explique aussi pourquoi la méthode est capable de traiter des milliers de feuilles dans un temps acceptable.

D'autre part, pour palier à l'incapacité de la méthode exacte de résoudre des instances de grandes dimensions, compte tenu de la combinatoire explosive du problème $MOST$, nous avons jugé utile de proposer notre méthode approchée GA-VNS. Cette dernière présente de bonnes caractéristiques d'après les résultats de l'expérimentation de part : sa capacité de calculer à la fois les solutions supportées et non supportées ; sa possibilité d'explorer des régions de l'espace de recherche que d'autres algorithmes approchés ne font pas, ainsi que la bonne répartition du front Pareto. En effet, les résultats de l'expérimentation que nous avons réalisés, montrent que le front de Pareto approché épouse bien la forme du front de Pareto exacte, puisque près de 70% des solutions non dominées obtenues par l'algorithme GA-VNS sont données par MOST-algorithme.

Comme futures pistes de recherche, nous citons la construction des arbres non dominés basée sur des outils de graphes ainsi que la réalisation d'un schéma d'hybridation d'autres methaheuristiques.

Bibliographie

[1] M. Abbas, M.E-A. Chergui et M. Ait Mehdi, Efficient cuts for generating the nondominated vectors for multiobjective integer linear programming, Int. J. Mathematics in Operational Research. Vol 4, No. 3 (2012).

[2] K.A. Andersen, K. Joernsten et M. Lind, On bicriterion minimal spanning trees : an approximation. Computers and Operations Research 23, 1171-1182 (1996).

[3] J.E.C. Arroyo, P.S. Vieira et D.S. Vianna, A GRASP algorithm for the multi-criteria minimum spanning tree problem. In : Second Brazilian Symp. on Graphs (GRACO 2005), Rio de Janeiro, Brazil, Also the Sec. Multidiscip. Conf. on Sched. : Theory and Apps., New York (2005).

[4] C. Berge, Graphes et hypergraphes, Dunod, Paris, (1970).

[5] J.A. Bondy, U.S.R. Murty, Graph Theory with Applications, North-Holland, New-York, (1976).

[6] A. Boumesbah, M.E-A. Chergui, AN EXACT METHOD TO GENERATE ALL NONDOMINATED SPANNING TREES, RAIRO Operations Research, DOI : 10.1051/ro/2016060, 50, 857-867 (2016).

[7] A. Boumesbah, M.E-A. Chergui, An exact method to solve the multiobjective minimum spanning tree problem, Les Annales RECITS (http ://www.lrecits.usthb.dz) Vol. 2, 1-6 (2015).

[8] A. Boumesbah, M.E-A. Chergui, An approximation of the Pareto frontier for the multi-objective spanning tree problem, OPERATIONAL RESEARCH PRACTICE IN AFRICA (ORPA) (2015).

[9] A. Boumesbah, M.E-A. Chergui, An approximation of the Pareto frontier for the multi-objective spanning tree problem, Lebanese International Conference on Mathematics and Applications (LICMA) (2015).

[10] P. M. Camerini, G. Galbiati et F. Maffioli, The complexity of weighted multi-constrained spanning tree problems. Colloquium the Theory of Algorithms, Colloquium. L. Lovàsz,Ed. Amsterdam : North-Holland, 53-101 (1984).

[11] I. Charon, A. Germa et O. Hudry, Méthode d'optimisation combinatoire, Collection Pédagogique de Communication, Masson Paris, ISBN-2-225-85307-X, (1996).

[12] G. Chen, S. Chen, W. Guo, H. Chen, The multi-criteria minimum spanning tree problem based genetic algorithm. Inf. Sci. 177, 5050-5063 (2007).

[13] N. Christofides, Graph Theory, An Algorithmic Approach. Academic Press Inc., London (1975).

[14] J. C. N. Clìmaco, E, A. Martins, bicriterion shortest path algorithm. European Journal of Operational Research. 11, 399-404 (1982) .

[15] Y. Collecte et P. Siarry, Optimisation Multi-objectif, Eyrolles, ISBN-2-212-11168-1 (2002).

[16] S. A. Cook The complexity of theorem proving procedures. In stoc71, 151-158 (1971).

[17] H. W. Corley, Efficient spanning trees, Craveirinha et M. Pascoal, Multicriteria routing models in telecommunication networks-overview and a case study. J. Optim, Theory Appl. 45, 481-485 (1985).

[18] C. G. Da Silva et J. C. N. Clìmaco, A note on the computation of ordered supported nondominated solutions in the bi-criteria minimum spanning tree problems. Journal of telecommunications and information technologie, 11-15 (2007).

[19] R. Dakin, *A tree search algorithm for mixed integer programming problems*, Computer Journal 8, pp. 250-255 (1965).

[20] G.B. Dantzig, Linear programming and extentions, Princeton University Press, (1963).

[21] M. David-Moradkhan et W. Browne, A hybridized evolutionary algorithm for multicriterion minimum spanning tree problem, In : Proc. 8th. Int. Conf. Hybrid Intell. Sys. (HIS 2008), pp. 290-295 (2008).

[22] M. David-Moradkhan et W. Browne, Evolutionary Algorithms for the Multi Criterion Minimum Spanning Tree Problem, Y. Tenne and C.-K. Goh (Eds.) : Computational Intel. in Expensive Opti. Prob., ALO 2, 423-452. (2010).

[23] K. Deb, S. Agrawal, A. Pratap, et T. Meyarivan, A fast and elitist multi-objective genetic algorithm for multi-objective optimization. *IEEE Trans. Evol.Comut. 6*, 181-197 (2002).

[24] F.Y. Edgeworth, Mathematical Physics. P. Keagan, London, (1881).

[25] M. Ehrgott Multicriteria Optimization, chap. 7, Lecture Notes in Economics and Mathematical Systems, pp. 153-222 (2000).

[26] M. Ehrgott et X. Gandibleux, A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spektrum 22, 425-460 (2000).

[27] M. Ehrgott, H.W. Hamacher, k. Klamroth, Nickels., A. Schobel et M.M. Weicek, A note on the equivalence of balance points and Pareto solutions in multiple-objective programming, Journal of Optimization theory and Applications, Vol.92, nol,209-212 (1997).

[28] M. Ehrgott, Multicriteria Optimization, 2nd edn. Springer, Berlin (2005).

[29] P. Erdös et A. Rényi, On the evolution of random graphs, Publications of the Mathematical Institute of the Hungarian Academy of Sciences. 5 (1960).

[30] R. Faure, Précis de recherche opérationnelle, Dunod Paris, ISBN-2-04-011029-1, 1979.

[31] J.C. Fournier. Graphe et application 1, Informatique et Système d'Information, Lavoisier. ISBN général 978-2-7462-1556-6, ISBN volume 978-2-7462-1656-3, (2007).

[32] H.N. Gabow, Two algorithms for generating weighted spanning trees in order. SIAM Journal on Computing 6, 139-150 (1977).

[33] L. Galand, Méthodes exactes pour l'optimisation multicritère dans les graphes : recherche de solution de compromis. PhD thesis, Université de Paris VI, (2008).

[34] A.M. Geoffrion, Proper efficiency and the theory of vector maximization,Journal of Mathematical Analysis and Applications, 22, pp. 618-630 (1968).

[35] J. Gottlieb, B.A. Julstrom, F. Rothlauf, G.R. Raidl, Prufer numbers : A poor representation of spanning trees for evolutionary search, in : Proceedings of the Genetic

and Evolutionary Computation Conference, Morgan Kaufmann, San Francisco, USA, 343 ?350 (2001).

[36] C. Guéret, C. Prins et M. Sevaux, Programmation linéaire, 65 problèmes d'optimisation modélisés et résolus avec visual Xpress, Eyrolles, ISBN : 2-212-09202-4 (2000).

[37] W. Guo, G. Chen et X. Feng, L. Yu, : Solving multi-criteria minimum spanning tree problem with discrete particle swarm optimization. In : Proc. 3rd Int. Conf. Nat. Comput. (ICNC 2007), pp. 471-478 (2007).

[38] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical Society 64, pages 275-278 (1958).

[39] Y. Haimes, L. Lasdon et D. Wismer, On a bicriterion formulation of the problems of integrated system identification and system optimization, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 1, pp.296-297 (1971).

[40] H. W. Hamacher et G. Ruhe, On spanning tree problems with multiple objectives. Anals of Operations Research. 52, 209-230 (1994) .

[41] L. Han et Y. Wang, A novel genetic algorithm for multi-criteria minimum spanning tree problem. In : Hao, Y., Liu, J., Wang, Y.-P., Cheung, Y.-m., Yin, H., Jiao, L., Ma, J., Jiao, Y.-C. (eds.) CIS 2005. LNCS (LNAI), vol. 3801, Springer, Heidelberg, 297-302 (2005).

[42] P. Hansen et N. Mladenović, Variable neighborhood search. Computers and Operations Research. 24 (11), 1097-1100 (1997).

[43] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica. 4, 373-395 (1984).

[44] L. Khachiyan, A polynomial algorithm in linear programming, Soviet Mathematics Doklady. 20, 191-194 (1979).

[45] T. Kavitha, C. Liebchen, K. Mehlhorn, D. Michaild, R. Rizzie, T. Ueckerdtf et K.A. Zweig, Survey Cycle bases in graphs characterization, algorithms, complexity, and applications , Computer Science Review Volume 3, Issue 4, November, 199-243 (2009).

[46] J. Knowles et D. Corne, Enumeration of pareto optimal multi-criteria spanning trees - a proof of the incorrectness of Zhou and Gen's proposed algorithm, European Journal of Operational Research, tm. 143, pp. 543-547 (2002).

[47] J. B. Kruskal, On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem. Proceedings of the American Mathematical Society. 7, 48-50 (1956).

[48] H. Land et A.G. Doig, An automatic method for solving discrete programming problems. Econometrica,28,497-520 (1960).

[49] C. W. Marshall, Applied graph theory. Wiley-Interscience. (1971).

[50] M. Özlen et M. Azizoğlu, Multi-objective integer programming, A general approach for generating all nondominated solutions. European Journal of Operational Research. 199, 25-35 (2009).

[51] M. Özlen, B. A. Burton et C. A. G. MacRae, Multi-objective integer programming : An Improved Recursive Algorithm. J. Optim, Theory Appl. 160, 470-482 (2014).

[52] V. Pareto, Cours d'économie politique.Rouge, Lausanne, (1896).

[53] M. Pirlot et J. Teghem, Résolution de problèmes de RO par les métaheuristiques, Lavoisier, ISBN-2-7462-0463-0, (2003).

[54] R. C. Prim, Shortest Connection Networks and Some Generalizations. Bell system Technical Journal. 36, 1389-1401 (1957).

[55] R. Ramos, S. Alonso, J. Sicilia et C. Gonzalez , The problem of the optimal biobjective spanning tree, European Journal of Operational Research, tm. 111, pp. 617-628 (1998).

[56] S.S. Rao, *Engineering optimization : theory and practice* , Wiley, John & Sons, Incorporated, (1996).

[57] M. Sakarovitch, *Optimisation combinatoire. Méthodes mathématiques et algorithmiques. Graphes et Programmation Linéaire*, (1984).

[58] M. Sakarovitch, *Optimisation combinatoire. Méthodes mathématiques et algorithmiques. Programmation Discrète*, Paris, Hermann, (1984).

[59] A. Schrijver, Theory of Linear and Integer Programming, Wiley and Sons, (1986) .

[60] P. Serafni, Some considerations about computational complexity for multi-objective combinatorial problems, dans J. Jahn et W. Krabs, réds., Recent advances and historical development of vector optimization, tm. 294 de Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, (1986).

[61] M. Simonnard, *Programmation linéaire : technique du calcul économique*, Paris, (1972).

[62] F. Sourd et O. Spanjaard, multi-objective branch-and-bound framework. Application to the bi-objective spanning tree problem. INFORMS Journal of Computing. 20(3), 472–484 (2008).

[63] O. Spanjaard, Exploitation des préférences non-classiques dans les problèmes combinatoires : modèles et algorithmes pour les graphes., Thèse de doctorat, Université Paris-Dauphine (Paris 9), (2003).

[64] N. Srinivas et K. Deb, Multi-Objective Function Optimization Using Non-Dominated Sorting Genetic Algorithm. Evol. Comput. 2, 221-248 (1995).

[65] S. Steiner and T. Radzik, Computing all efficient solutions of the biobjective minimum spanning tree problem. Computers & Operations Research. 35, 198-211(2008).

[66] R. E. Steuer, Multiple criteria optimization : theory, computation, and application. Wiley Series in Probability and Mathematical Statistics - Applied, Wiley. (1986).

[67] H.A. Taha, *Integer programming theory, applications and computations*, Academic Press, New York, (1975).

[68] J. Teghem, Gestion de Production, Méthodes Aléatoires, Aide Multicritère, Ellipes, ISBN-978-2-7289-8202 0, (2013).

[69] E.L. Ulungu et J. Teghem, The two phases méthod : An efficient procedure to solve bi-objective combinatorial optimization problems. Foundations of computing and Decision Sciences, 20 :149-165 (1995).

[70] E.L. Ulungu, Optimisation combinatoire multi-critère, Détérmination de l'ensemble des solutions efficaces et méthodes intéractives. PhD thesis, Université de Mons-Hainault, Faculté des sciences, (1993).

[71] T. Vincent, Caractérisation des solutions efficaces et algorithmes d'énumération exacts pour l'optimisation multi-objectif en variables mixtes binaires : PhD thesis, UNIVERSITÉ DE NANTES, (2013).

[72] G. Zhou et M. Gen, Genetic algorithm approach on multi-criteria minimum spanning tree problem, European Journal of Operational Research, tm. 114, pp. 141-152 (1999).

[73] E. Zitzler et L. Thiele, Multiobjective Evolutionary Algorithm : A Comparative Case Study and the Strength Pareto Approach. IEEE Trans. On Evolutionary Computation. 4, 257-271 (1999).