Élicitation incrémentale combinée à la recherche heuristique pour l'optimisation combinatoire multi-objectifs

Cassandre Leroy



Sorbonne Université (UFR d'Ingénierie)
École Doctorale Informatique, Télécommunications et Electronique
LIP6 (Équipe DECISION)

Thèse de doctorat en Informatique
Spécialité : aide à la décision

Élicitation Incrémentale combinée à la Recherche Heuristique pour l'optimisation combinatoire multi-objectifs

---

Soutenue le... 2022 par **Cassandre Leroy** devant le jury composé de :

Rapporteurs :	Matthieu Basseur Professeur, Université du Littoral
	Laëtitia Jourdan Professeure, Université de Lille
Examinateur :	Daniel Vanderpooten Professeur, Université Paris-Dauphine
Directeur :	Patrice Perny Professeur, Sorbonne Université
Encadrants :	Nawal Benabbou Maîtresse de conférences, Sorbonne Université
	Thibaut Lust Maître de conférences, Sorbonne Université

Table des matières

Introduction

La décision multicritère est un domaine de recherche qui étudie des problèmes de décision tenant compte de plusieurs critères à la fois, traduisant les différents objectifs d'un individu (appelé décideur). Ce type de problème représente la majorité des décisions que nous prenons, des plus simples aux plus complexes, de l'achat d'un produit au supermarché aux stratégies commerciales des plus grandes entreprises. L'assistance lors de ces décisions est rendue possible grâce aux outils informatiques, permettant notamment de faciliter la prise de décision mais aussi l'exploration et la compréhension des différentes solutions du problème.

La caractérisation de la solution optimale, c'est-à-dire la solution la plus satisfaisante pour le décideur, est relativement complexe dans le cadre de problèmes de décision multicritère. En effet, les critères considérés sont très souvent conflictuels et il n'est pas toujours simple de déterminer la solution qui réalise le meilleur compromis aux yeux du décideur. C'est quelque chose que nous pouvons observer dans le cadre de problèmes simples, par exemple lorsqu'un client achète une voiture représentant pour lui le meilleur compromis entre l'esthétique du véhicule, son prix et son caractère polluant. Dans les problèmes de décision impliquant un très grand nombre d'alternatives possibles, il peut être difficile pour le décideur de déterminer la solution qu'il préfère. Dans la littérature, de nombreux modèles décisionnels ont été proposés pour représenter les préférences d'un décideur dans des situations de décision multicritère et guider l'exploration des alternatives. Parmi ces modèles, nous pouvons mentionner la somme pondérée, l'opérateur OWA [Yager, 1988], l'agrégateur WOWA [Torra, 1997], et l'intégrale de Choquet [Grabisch and Labreuche, 2010], qui sont des fonctions permettant d'associer à chaque alternative du problème une performance globale obtenue par agrégation de ses évaluations sur les différents critères.

L'élicitation des préférences est une problématique de recherche visant à concevoir des méthodes permettant d'aider le décideur dans sa prise de décision en collectant des informations sur ses préférences, par exemple, par le biais de questions-réponses. L'approche classique considère en entrée une base de données contenant des informations sur les préférences du décideur et consiste à déterminer les paramètres de la fonction d'agrégation qui représentent le mieux cette base de données ; à titre d'exemple, la méthode UTA permet d'éliciter les paramètres d'une fonction d'utilité additive en utilisant la programmation linéaire [Jacquet-Lagrèze and Siskos, 1982, Siskos and Yannacopoulos, 1985]. L'efficacité de cette approche dépend de la qualité de la base de données, et de sa taille. Il faudrait une très grande base de données pour parvenir à déterminer précisément les paramètres représentant au mieux les préférences du décideur. Cependant, il est souvent difficile d'obtenir une base

d'exemples de préférences conséquente en pratique, parce que le nombre de questions que l'on peut poser au décideur est limité. Afin de réduire le nombre d'interactions avec le décideur, des méthodes d'élicitation partielles ont été proposées, permettant de déterminer la meilleure solution pour le décideur sans chercher à apprendre précisément les paramètres du modèle [White et al., 1984]. Dans cette idée, une approche dite "incrémentale" consiste à poser des questions au décideur de manière itérative, de sorte à réduire efficacement l'espace des paramètres admissibles, jusqu'à être en mesure d'identifier la solution optimale. Plus récemment, il a été proposé d'utiliser le critère de décision minimax regret pour choisir les questions à poser au décideur afin de formuler rapidement une recommandation [Boutilier et al., 2006]. Notre objectif est d'étendre efficacement cette approche d'élicitation à des problèmes de décision sur domaine combinatoire.

La conception de procédures d'élicitation efficaces sur domaine combinatoire est l'un des sujets d'actualité de la théorie de la décision algorithmique. Déterminer par élicitation la meilleure solution parmi un très grand nombre est une tâche encore plus complexe, ce qui a conduit de nombreux chercheurs à proposer des méthodes d'élicitation dans divers domaines, comme dans des systèmes multi-agents [Benabbou and Perny, 2016, Bourdache and Perny, 2019] et dans des problèmes d'optimisation combinatoire multi-objectifs [Kaddani et al., 2017, Korhonen, 2005]. En particulier, une approche hybride consistant à entremêler la résolution exacte du problème et l'élicitation incrémentale fondée sur le minimax regret a été récemment proposée [Benabbou, 2017]. Il s'agit de poser des questions durant l'exploration des solutions du problème, à des moments précis, ce qui permet de discriminer les solutions rencontrées durant la résolution en réduisant l'imprécision sur les paramètres du modèle décisionnel en cours de recherche. Cette approche hybride a conduit à la conception d'algorithmes de résolution exactes et constructifs, pour des problèmes d'optimisation définis avec des modèles décisionnels très simples (linéaires).

Positionnement de la thèse

Notre premier objectif est d'étendre efficacement l'approche de résolution hybride fondée sur le minimax regret à des problèmes d'optimisation définis avec des modèles décisionnels plus complexes (par exemple, l'intégrale de Choquet). Pour ce faire, nous proposons une nouvelle approche consistant à utiliser des métaheuristiques (par exemple, l'approche évolutionnaire), au lieu d'algorithmes constructifs. Cette nouvelle approche permet de résoudre de manière approchée des problèmes d'optimisation combinatoire basés sur des préférences pour lesquels aucun algorithme exact efficace n'est connu, avec un nombre de questions relativement faible en pratique.

Notre second objectif est d'étudier le potentiel de cette approche pour l'optimisation dans les matroïdes. Un matroïde est une structure combinatoire classique permettant la modélisation de nombreux problèmes et leur résolution efficace. Dans ce cadre, nous proposons des algorithmes de recherche gloutonne et de recherche locale permettant de retourner des solutions (quasi-)optimales, en posant peu de questions au décideur empiriquement.

La classe des problèmes concernés inclut une large variété de problèmes d'optimisation

concrets tels que le problème du voyageur de commerce, de couverture maximale, d'arbre couvrant de poids minimum, de sac à dos, ...

Organisation de la thèse

Cette thèse se décompose en 4 chapitres. Le premier chapitre introduit les notions de base utile à la compréhension de cette thèse et présente le domaine de l'optimisation combinatoire multi-objectifs. Dans un premier temps, on décrit les méthodes de résolution exactes et approchées utilisées classiquement dans ce domaine. Dans un deuxième temps, on s'intéresse à la modélisation de préférences complexes en décision multicritère, en présentant différentes fonctions d'agrégation ainsi que les différentes grandes approches fondées sur des préférences pour la résolution de problèmes d'optimisation (méthodes a priori, a posteriori et interactives). Dans un troisième temps, on se concentre sur l'élicitation incrémentale des préférences, en étudiant des algorithmes issus de l'état de l'art, afin de mieux positionner nos travaux.

Le deuxième chapitre propose deux nouvelles approches combinant élicitation incrémentale et recherche heuristique : une méthode fondée sur la recherche locale et une inspirée de l'approche évolutionnaire. Dans un premier temps, on présente ces approches de manière générale, pour pouvoir les appliquer à n'importe quel problème d'optimisation combinatoire multi-objectifs, et on étudie leurs garanties de performances en termes de nombre de questions et temps de résolution. Dans un second temps, on montre comment adapter ces approches aux problèmes du voyageur de commerce et du sac à dos, avec des préférences représentées par une somme pondérée, un opérateur OWA et une intégrale de Choquet. Des résultats numériques sont fournis pour évaluer les performances empiriques de ces nouveaux algorithmes.

À partir du troisième chapitre, la thèse se concentre sur des problèmes qui se modélisent sous la forme de matroïde pondéré. Plus précisément, le troisième chapitre étudie des problèmes où les solutions sont évaluées par une fonction additive. Dans un premier temps, on présente des notions essentielles liées à théorie des matroïdes, et des algorithmes de résolution classiques dans le cadre de préférences précisément connues. Dans un second temps, on propose deux approches interactives pour la résolution exacte (ou approchée avec garantie) des problèmes avec préférences connues partiellement : une méthode gloutonne et une recherche locale. On étudie leurs garanties de performances en termes de nombre de questions, temps de résolution et qualité de la solution retournée. Ces algorithmes sont évalués expérimentalement sur trois problèmes classiques : arbre couvrant, ordonnancement et sélection de sous-ensembles.

Le quatrième chapitre étudie également des problèmes se modélisant sous forme de matroïde pondéré, mais avec une fonction d'évaluation sous-modulaire, afin de représenter des interactions possibles entre les éléments d'une solution. Dans un premier temps, on propose une adaptation des algorithmes interactifs du troisième chapitre pour le cas sous-modulaire, et on analyse leurs garanties de performances (nombre de questions, temps de résolution et qualité de la solution retournée). Dans un second temps, ces algorithmes sont testés sur des problèmes pratiques impliquant une structure de matroïde et une fonction sous-modulaire à

maximiser, comme le problème de couverture maximal ainsi que la sélection collective d'éléments. Nous introduirons également quelques résultats concernant la minimisation d'une fonction sous-modulaire.

Enfin le dernier chapitre conclut cette thèse par une synthèse des résultats obtenus et présente plusieurs perspectives de recherche pour la poursuite de ces travaux.

Chapitre 1

État de l'art

Résumé

Dans ce chapitre, nous commençons par les bases du domaine de l'optimisation combinatoire multi-objectifs, en présentant notamment la dominance de Pareto, le front de Pareto, les solutions efficaces (supportées ou non-supportées), ainsi que le point idéal et le point nadir. Nous présentons des méthodes classiques pour la détermination exacte du front de Pareto ou d'une bonne représentation (avec des métaheuristiques).

Dans une seconde partie, nous nous intéressons à la détermination d'une solution efficace de bon compromis, en présentant des fonctions d'agrégation classiques issues du domaine de l'aide à la décision multicritère. Nous nous intéressons plus particulièrement à la somme pondérée, l'opérateur OWA et aux intégrales de Choquet. Nous présentons les trois grandes approches pour déterminer la solution optimale, étant donné un modèle de préférences : les méthodes a priori, a posteriori et interactives.

Enfin, la dernière partie de ce chapitre porte sur des méthodes interactives particulières. Plus précisément, nous nous intéressons à l'élicitation incrémentale des préférences. Nous y décrivons des méthodes classiques, mais aussi des méthodes plus récentes basées sur le critère du minimax regret.

1.1 Optimisation combinatoire multi-objectifs

En informatique, comme en mathématiques et en économie, un problème d'optimisation est un problème qui consiste à trouver la meilleure solution parmi un ensemble de solutions donné (les "alternatives"). Dans un problème d'optimisation combinatoire, l'ensemble des solutions à considérer est discret et fini mais très grand. Il est dans ce cas défini de manière implicite comme étant l'ensemble des solutions vérifiant certaines propriétés (contraintes) liées au problème. En pratique, les alternatives d'un problème sont souvent comparées par le biais de plusieurs fonctions traduisant des objectifs à atteindre ; c'est pourquoi ces fonctions sont souvent appelées fonctions objectifs. Ces différents objectifs peuvent être liés à la prise en considération de différents points de vue comme les finances, les aspects environnementaux, les délais, la sécurité, la qualité, l'éthique et bien d'autres. Pour le choix d'un itinéraire par exemple, il peut y avoir un objectif de durée, de coût, ou encore environnemental (comme l'empreinte carbone). Citons d'autres exemples : la gestion de portefeuilles [Steuer et al., 2005], l'optimisation du rendement énergétique d'un parc éolien [Yin Kwong et al., 2014] ou encore l'organisation de voyages touristiques [Godart, 2001]. Tous ces exemples rentrent dans le domaine de l'optimisation combinatoire multi-objectifs. Dans ce contexte, une solution réalisable est un vecteur $x = (x_1, \ldots, x_m)$ de m variables appartenant à $\{0,1\}^m$ et devant vérifier q contraintes $c_i(x) \leq 0$, avec $i \in \{1, \ldots, q\}$. On notera $\mathcal{X}$ l'ensemble des solutions réalisables dans cette thèse. Étant donné $y : \{0,1\}^m \to \mathbb{Z}^p$ une fonction qui associe à tout $x \in \mathcal{X}$ un vecteur $y(x) = (y_1(x), \ldots, y_p(x))$ représentant les performances de x selon p objectifs, il s'agit de déterminer :

$$\begin{cases} \underset{x \in \{0,1\}^m}{\text{minimiser}} & y(x) = (y_1(x), \ldots, y_p(x)) \\ \text{s.c.} & c_i(x) \leq 0, \ \forall i \in \{1, \ldots, q\} \end{cases}$$

Dans cette thèse on considère que les fonctions objectifs sont à minimiser (représentant par exemple un coût, une durée), le cas de la maximisation pouvant se ramener à une minimisation par un simple changement de signe. L'image des solutions réalisables dans l'espace des objectifs sera notée $\mathcal{Y}$ dans la suite de ce document. L'image d'une solution dans l'espace des objectifs est communément appelée point.

Le mot combinatoire fait références à toutes les combinaisons possibles des valeurs prises par les variables du problème. La taille du problème (autrement dit le nombre de variables) joue un rôle essentiel ici car le nombre de solutions réalisables augmente de façon exponentielle avec la taille du problème [Karasakal and Köksalan, 2009]. Par exemple, pour un problème combinatoire où une solution réalisable correspond à une permutation de m variables, sans aucune autre contrainte, il existe $m!$ solutions admissibles. Et donc, même pour un petit problème de taille 20, on obtiendrait 20! solutions, ce qui correspond à 2.432902×10^{18} possibilités. À raison de 10^{-6} seconde pour évaluer une solution générée, il faudrait 771 siècles pour énumérer toutes les solutions, ce qui exclut les méthodes de résolution classiques basées sur l'énumération de toutes les solutions admissibles.

L'exemple ci-dessous permet d'illustrer les notions introduites jusqu'ici sur un petit problème d'optimisation combinatoire.

Exemple 1.1.1. *Dans la Figure 1.1(a), on représente l'ensemble des solutions réalisables pour un problème d'optimisation combinatoire avec $m = 3$ variables. À chaque solution réalisable, on associe un point dans l'espace des objectifs, défini par $p = 2$ fonctions objectifs et représenté dans la Figure 1.1(b).*

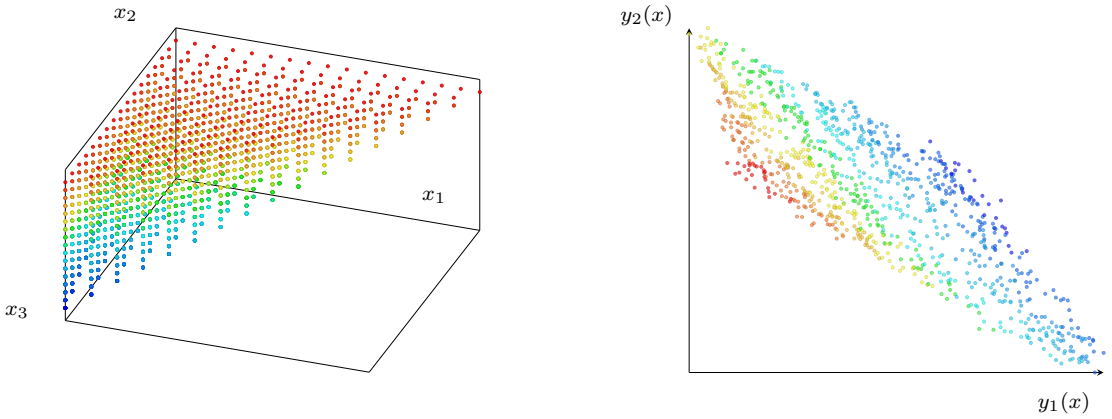


FIGURE 1.1 – Représentation de l'ensemble $\mathcal{X}$ des solutions réalisables ($m = 3$), et de $\mathcal{Y}$ son image dans l'espace objectif ($p = 2$).

On remarque que certains objectifs sont contradictoires, comme cela arrive fréquemment dans des problèmes d'optimisation multi-objectifs. Par exemple, dans l'espace des objectifs, lorsque le second objectif y_2 prend une valeur faible, alors on observe que y_1 prend quant à lui une valeur haute, et inversement. Si y_3 prend une valeur faible alors y_1 et y_2 ont des valeurs plutôt élevées.

En optimisation mono-objectif ($p = 1$), les solutions optimales sont celles qui minimisent la fonction objectif. En optimisation multi-objectifs, la notion de meilleure solution ne va de soi. Par exemple, lors d'un voyage Paris-Marseille en voiture, deux solutions principales se présentent : prendre l'autoroute ou prendre les voies secondaires. Pour minimiser la durée, il faudrait prendre l'autoroute mais cela augmente le coût du voyage (et l'inverse pour l'option des voies secondaires). De manière générale, dans un problème multi-objectifs, les différentes fonctions à optimiser sont souvent contradictoires. Il est alors essentiel de pouvoir comparer des vecteurs de performances afin de différencier une bonne solution d'une mauvaise. La dominance de Pareto est une réponse possible à ce besoin.

1.1.1 Dominance de Pareto et solutions efficaces

Avant de présenter la dominance de Pareto, il est nécessaire de définir la notion de relation binaire entre solutions.

Definition 1.1.1 (Relation binaire)**.** *Une relation binaire $\mathcal{R}$ sur un ensemble $\mathcal{X}$ est définie par un sous-ensemble $\mathcal{G}$ de $\mathcal{X} \times \mathcal{X}$. Si $(x, y) \in \mathcal{G}$, on dit alors que x est en relation avec y et on le note $x\mathcal{R}y$.*

Une relation binaire peut avoir plusieurs propriétés intéressantes dans le cadre de l'optimisation multi-objectifs, dont on rappelle ici les principales.

Definition 1.1.2. *Soit une relation binaire $\mathcal{R}$, on dit que $\mathcal{R}$ est :*
- *Réflexive si et seulement si $x\mathcal{R}x$ pour tout $x \in \mathcal{X}$.*
- *Irréflexive si et seulement si non$(x\mathcal{R}x)$ pour tout $x \in \mathcal{X}$, où non est l'opérateur de négation logique.*
- *Symétrique si et seulement si pour tout $x, y \in \mathcal{X}$, $x\mathcal{R}y$ implique $y\mathcal{R}x$.*
- *Asymétrique si et seulement si aucun couple $(x, y) \in \mathcal{X}^2$ ne vérifie $x\mathcal{R}y$ et $y\mathcal{R}x$ simultanément.*
- *Antisymétrique si et seulement si pour tout couple $(x, y) \in \mathcal{X}^2$, $x\mathcal{R}y$ et $y\mathcal{R}x$ impliquent $x = y$.*
- *Transitive si et seulement si pour tout $x, y \in \mathcal{X}$, $x\mathcal{R}y$ et $y\mathcal{R}z$ impliquent $x\mathcal{R}z$.*
- *Totale si et seulement si pour tout couple $(x, y) \in \mathcal{X}^2$, on a $x\mathcal{R}y$ ou $y\mathcal{R}x$.*

À l'aide de ces propriétés, on peut maintenant distinguer quatre types de relation binaire.

Definition 1.1.3. *On dit qu'une relation binaire $\mathcal{R}$ sur $\mathcal{X}$ est :*
- *Un préordre sur $\mathcal{X}$ si la relation $\mathcal{R}$ est réflexive et transitive.*
- *Une équivalence sur $\mathcal{X}$ si $\mathcal{R}$ est réflexive, transitive et symétrique.*
- *Un ordre sur $\mathcal{X}$ si $\mathcal{R}$ est réflexive, transitive et antisymétrique.*
- *Un ordre strict sur $\mathcal{X}$ si $\mathcal{R}$ est asymétrique et transitive.*

La dominance de Pareto est une relation binaire qui peut permettre de comparer deux solutions entre elles sans avoir besoin d'information supplémentaire sur le problème.

Definition 1.1.4 (Pareto-dominances)**.** *On distingue trois dominances au sens de Pareto :*
- *Pareto-dominance faible : Le point $a = (a_1, \ldots, a_p) \in \mathbb{R}^p$ Pareto-domine faiblement le point $b = (b_1, \ldots, b_p) \in \mathbb{R}^p$ (noté $a \precsim_P b$) si et seulement si $a_i \leq b_i$ pour tout $i \in \{1, \ldots, p\}$.*
- *Pareto-dominance : Le point $a = (a_1, \ldots, a_p) \in \mathbb{R}^p$* Pareto-domine *le point $b = (b_1, \ldots, b_p) \in \mathbb{R}^p$ (noté $a \prec_P b$) si et seulement si $a_i \leq b_i$ pour tout $i \in \{1, \ldots, p\}$ et $a_i < b_i$ pour au moins une valeur $i \in \{1, \ldots, p\}$.*
- *Pareto-dominance stricte : Le point $a = (a_1, \ldots, a_p) \in \mathbb{R}^p$ Pareto-domine strictement le point $b = (b_1, \ldots, b_p) \in \mathbb{R}^p$ (noté $a \ll_P b$) si et seulement si $a_i < b_i$ pour tout $i \in \{1, \ldots, p\}$.*

La relation $\precsim_P$ est réflexive, transitive et antisymétrique, tandis que les relations $\prec_P$ et $\ll_p$ sont irréflexives, transitives et asymétriques. Il existe des relations entre ces différentes définitions : la dominance stricte implique la dominance et la dominance implique la dominance faible. Aussi, aucune de ces définitions n'induit un ordre total, ce qui signifie que certaines paires de vecteurs peuvent être incomparables avec la relation de dominance de Pareto. Ceci nous conduit à définir la notion d'optimalité pour les dominances de Pareto.

Definition 1.1.5 (Solution et ensemble efficace)**.** *Une solution $x \in \mathcal{X}$ est dite efficace s'il n'existe pas de solution réalisable $x' \in \mathcal{X}$ telle que $y(x') \prec_P y(x)$. Pour un problème, l'ensemble efficace est l'ensemble des solutions efficaces, noté $\mathcal{X}_E$.*

Definition 1.1.6 (Solution et ensemble faiblement efficace)**.** *Une solution $x \in \mathcal{X}$ est dite faiblement efficace s'il n'existe pas de solution réalisable $x' \in \mathcal{X}$ telle que $y(x) \ll_p y(x')$. L'ensemble faiblement efficace est l'ensemble contenant toutes les solutions faiblement efficaces d'un problème donné.*

En d'autres termes, cela signifie que pour être une solution efficace, aucune autre solution ne doit la dominer. Par contre, une solution faiblement efficace peut être dominée par une autre, mais pas strictement. L'image d'une solution efficace dans l'espace des objectifs est appelée point non-dominé (ou non Pareto-dominé) tandis que l'image d'une solution faiblement efficace est appelée point faiblement non-dominé (ou non Pareto-faiblement dominé). En optimisation multi-objectifs on s'intéresse à déterminer le front de Pareto défini comme suit :

Definition 1.1.7 (Front de Pareto)**.** *L'image de l'ensemble efficace $\mathcal{X}_E$ dans l'espace des objectifs est appelé front de Pareto, noté $\mathcal{Y}_N$.*

Pour illustrer ces différentes notions, on s'intéresse à l'exemple suivant :

Exemple 1.1.2. *On considère un problème bi-objectifs dont les solutions sont représentées en Figure 1.2. Par exemple, le point b est Pareto dominé strictement par le point a. En effet, le point a est strictement plus performant que le point b sur chacun des deux objectifs. Le point b' est quant à lui simplement Pareto dominé par le point a' car ce dernier fait mieux que b' sur le premier objectif (et est au moins aussi bon sur le second).*

Graphiquement, il est facile de voir que les points en gris sont dominés au sens de Pareto puisqu'ils sont situés dans le cône de dominance (zone rectangulaire grise) d'un autre point. Les points en noir constituent donc le front de Pareto et correspondent aux solutions efficaces du problème. Le point b' est, quant à lui, faiblement Pareto-dominé car aucune solution n'est meilleure sur tous les objectifs. Ainsi, l'ensemble des solutions faiblement efficaces est composé de l'ensemble efficace et de la solution associée au point b'.

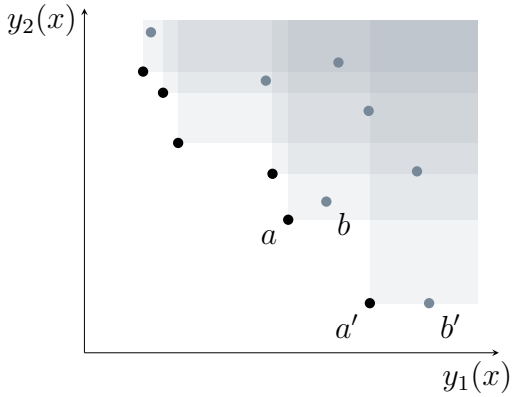


FIGURE 1.2 – Représentation de l'image des solutions d'un problème bi-objectifs.

Il existe une classification importante des problèmes d'optimisation combinatoire multi-objectifs basée sur la notion de convexité ou non convexité du front de Pareto. Cette classification joue un rôle important dans le choix de la méthode de résolution, ce que nous verrons dans la section suivante (Section 1.1.2).

Definition 1.1.8 (Ensemble convexe). *Un ensemble $\mathcal{Y}$ est convexe si et seulement si, $\forall A, B \in \mathcal{Y}$, le segment $[A, B]$ qui joint ces deux points est entièrement contenu dans l'ensemble $\mathcal{Y}$.*

L'introduction de la convexité nous permet une nouvelle classification des solutions efficaces dans l'espace des objectifs. On peut maintenant différencier les solutions efficaces supportées des non-supportées.

Definition 1.1.9 (Solution efficace supportée). *Soit $x \in \mathcal{X}_E$ une solution efficace. S'il existe $\lambda \in \mathbb{R}^p_>$ tel que x est une solution optimale pour $\min_{x \in \mathcal{X}} \sum_{i=1}^{p} \lambda_i f_i(x)$ alors x est appelé solution efficace supportée.*

L'image dans l'espace des objectifs d'une telle solution est appelée point non-dominé supporté. Ces points sont sur les bords de l'enveloppe convexe étendue de $\mathcal{Y}$, soit formellement $\text{conv}(\mathcal{Y}) + \mathbb{R}^p_\geq$ avec $\mathbb{R}^p_\geq$ l'orthant non négatif de $\mathbb{R}^p$. Ainsi l'enveloppe étendue correspond à l'union de tous les cônes qui pointe sur une solution de l'enveloppe convexe. De plus, lorsque l'image des solutions réalisables dans l'espace des objectifs est convexe, l'ensemble des solutions efficaces est égal à l'ensemble des solutions efficaces supportées.

Definition 1.1.10 (Solution efficace non-supportée). *Une solution efficace est dite non-supportée si son image dans l'espace des objectifs est située à l'intérieur de l'enveloppe convexe étendue de $\mathcal{Y}$.*

Une notion très utile en optimisation multi-objectifs qui permet aussi d'évaluer le front de Pareto est la notion de point de référence. Nous verrons dans la Section 1.1.3 comment ces points permettent d'obtenir ou approcher le front de Pareto. En effet, les valeurs prisent sur chaque objectif par les points non dominés sont délimitées par deux points de références fictifs : le point idéal et le point nadir.

Definition 1.1.11 (Point idéal). *Le point idéal $z^* = (z_1^*, \ldots, z_p^*)$ est défini par les meilleures performances possibles sur chacun des objectifs de la manière suivante :*

$$z_i^* = \min_{x \in \mathcal{X}_E} y_i(x)$$

Definition 1.1.12 (Point nadir). *Le point nadir $\hat{z} = (\hat{z}_1, \ldots, \hat{z}_p)$ est défini par les pires performances atteintes sur chacun des objectifs par les solutions efficaces :*

$$\hat{z}_i = \max_{x \in \mathcal{X}_E} y_i(x)$$

On remarque que pour déterminer le point idéal on n'a pas besoin de connaître l'ensemble $\mathcal{X}_E$ car le minimum sur un critères i obtenu grâce à l'ensemble $\mathcal{X}_E$ est aussi le minimum obtenu grâce à l'ensemble $\mathcal{X}$. En revanche, ce n'est pas le cas pour le point nadir, d'où une plus grande difficulté de le déterminer.

Si on considère les pires performances possibles sur l'ensemble des solutions réalisables, c'est-à-dire sans le contraindre aux solutions efficaces, alors ce point est appelé point anti-idéal. On ajoute que l'on peut définir de la même façon un point idéal local et un point nadir local, c'est-à-dire défini sur un sous-ensemble $\mathcal{X}' \subseteq \mathcal{X}$. Ces différentes définitions sont illustrées dans l'exemple suivant.

Exemple 1.1.3. *Sur l'exemple de la Figure 1.3, la zone grisée représente l'enveloppe convexe de l'ensemble des points $\mathcal{Y}$ et son extension, soit conv($\mathcal{Y}$) + $\mathbb{R}^p_{\geq}$. Les points en bleus sont situés à l'intérieur de l'enveloppe convexe étendue, les solutions correspondantes sont donc des solutions efficaces non supportées. Les points en noirs sont situés sur le bord de l'enveloppe convexe étendue, ce sont donc des points correspondants à des solutions efficaces supportés.*

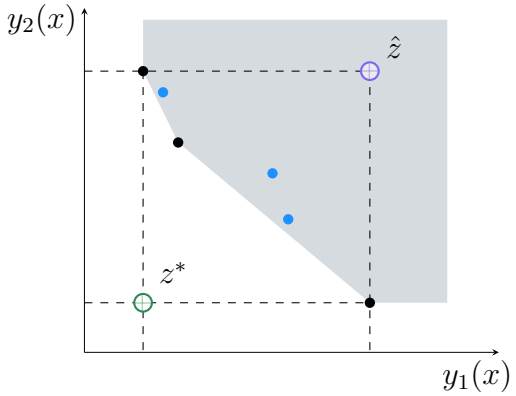


FIGURE 1.3 – Enveloppe convexe étendue des solutions efficaces d'un problème bi-objectifs, et représentation des points idéal et nadir.

Déterminer l'ensemble des solutions efficaces ou tester si une solution est efficace sont des tâches compliquées en optimisation combinatoire multi-objectifs, intéressons nous maintenant à une méthode de classification prenant en compte cette difficulté.

1.1.2 Complexité et intraitabilité

Dans cette section, nous allons parler de la complexité et de l'intraitabilité (infaisabilité) des problèmes d'optimisation combinatoire multi-objectifs. La difficulté de générer l'ensemble efficace provient essentiellement de ces deux problèmes :

— Il est difficile de vérifier si une solution est efficace (complexité).

— Le nombre de points non dominés peut être très grand (intraitabilité).

La théorie de la complexité est un vaste domaine qui étudie la difficulté de fournir des algorithmes efficaces pour des problèmes généraux et spécifiques. Cette difficulté est mesurée

par le nombre d'opérations dont un algorithme a besoin, dans le pire des cas (analyse du pire des cas), pour trouver la réponse correcte au problème de décision considéré. Un problème de décision est une question qui ne peut avoir que deux réponses : "oui" ou "non". Le problème de décision associé à un problème de minimisation mono-objectif est la question suivante : étant donné une constante $b \in \mathbb{Z}$, existe-t-il une solution $x \in \mathcal{X}$ telle que $y(x) \leq b$? Pour un problème d'optimisation multi-objectifs, le problème de décision associé consiste à décider, étant donné un vecteur d'entiers $b = (b_1, \ldots, b_p)$, s'il existe une solution réalisable telle que $y_i(x) \leq b_i$ pour tout $i \in \{1, \ldots, p\}$.

Il existe une grande variété de classes de complexité dans lesquelles on peut ranger les problèmes de décision.

Definition 1.1.13 (Classe $\mathcal{P}$)**.** *Un problème de décision appartient à la classe $\mathcal{P}$ des problèmes, s'il existe un algorithme déterministe qui répond au problème de décision et nécessite $\mathcal{O}(g(m))$ opérations, où g est un polynôme en m et m la taille de l'entrée du problème.*

On considère que ces problèmes sont faciles à résoudre.

Definition 1.1.14 (Classe $\mathcal{NP}$)**.** *Un problème de décision appartient à la classe $\mathcal{NP}$ s'il peut être décidé par une machine de Turing non déterministe en temps polynomial par rapport à la taille de l'entrée m.*

Remarquons que $\mathcal{NP}$ ne veut pas dire non polynomial mais non déterministe polynomial. Un algorithme non déterministe, par opposition à un algorithme déterministe, peut avoir un comportement différent sur plusieurs exécutions de la même instance du problème. C'est pourquoi, il peut nécessiter un nombre polynomial d'opérations sur une exécution alors qu'une autre exécution nécessitera un nombre exponentiel. On peut aussi dire que la classe $\mathcal{NP}$ contient l'ensemble des problèmes dont on peut vérifier une solution en temps polynomial. Ainsi, la classe $\mathcal{P}$, des problèmes faciles à résoudre, est nécessairement contenue dans la classe $\mathcal{NP}$ des problèmes faciles à vérifier. On dit aussi des problèmes dans $\mathcal{NP}$ qu'ils sont difficiles à résoudre, même si pour certains d'entre eux, des algorithmes très efficaces en pratique existent (pour plus de détails, voir [Garey, 1979, Teghem, 2003]).

Definition 1.1.15 ($\mathcal{NP}$-complet)**.** *Un problème de décision P est $\mathcal{NP}$-complet si $P \in \mathcal{NP}$ et pour tout $P' \in \mathcal{NP}$, il existe une transformation polynomiale de P' vers P permettant de résoudre P' en temps polynomial.*

On peut résumer ces définitions dans la Figure 1.4 dans laquelle on suppose que $\mathcal{P} \neq \mathcal{NP}$.

Definition 1.1.16 ($\mathcal{NP}$-difficile)**.** *Un problème d'optimisation est $\mathcal{NP}$-difficile s'il est au moins aussi difficile qu'un problème $\mathcal{NP}$-complet.*

Ainsi, les problèmes d'optimisation dont les problèmes de décision sont $\mathcal{NP}$-complets sont $\mathcal{NP}$-difficiles, mais ce ne sont pas les seuls problèmes $\mathcal{NP}$-difficiles. Il existe des sous-classes de complexité pour lesquelles un changement de classification d'un de ses problèmes induirait un bouleversement majeur dans la théorie de la complexité. La question "L'ensemble $\mathcal{P}$ est égal à l'ensemble $\mathcal{NP}$?" est toujours ouverte aujourd'hui. Si un problème $\mathcal{NP}$-complet

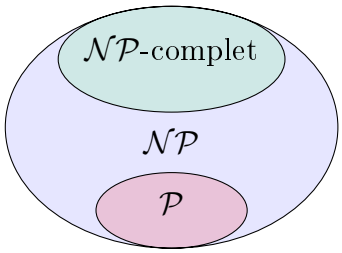


FIGURE 1.4 – Illustration de la complexité des problèmes d'optimisation.

ou $\mathcal{NP}$-difficile était résolu par un algorithme déterministe polynomial, cela impliquerait nécessairement que "$\mathcal{P} = \mathcal{NP}$" car résoudre en temps polynomial un problème $\mathcal{NP}$-complet permet de résoudre tous les problèmes de NP [Garey, 1979].

Remarquons aussi que certains problèmes combinatoires pour lesquels il existe un algorithme polynomial dans leur version mono-objectif sont $\mathcal{NP}$-difficiles dans leur version multi-objectifs. Par exemple, c'est le cas du problème de l'arbre de poids minimal que l'on étudiera dans le chapitre 3.4. Dans le chapitre 2.3, on s'intéressera aux problèmes du sac à dos et du voyageur de commerce, qui sont des problèmes $\mathcal{NP}$-difficiles dans leur version mono-objectif et qui le restent évidemment dans leurs équivalents multi-objectifs.

Concentrons-nous maintenant sur la notion d'intraitabilité d'un problème [Ehrgott, 2005].

Definition 1.1.17 (Intraitabilité)**.** *Un problème d'optimisation multicritère est dit intraitable (infaisable) si le nombre de points non-dominés peut être exponentiel en la taille du problème.*

La plupart des problèmes classiques sont intraitables, par exemple l'arbre couvrant de poids minimal et le voyageur de commerce.

Dans la littérature, les problèmes d'optimisation combinatoires multi-objectifs sont traités de différentes manières : trouver une solution efficace par point non dominé (ou une bonne approximation) ou trouver une solution de bon compromis selon les préférences subjectives d'une personne. Pour les problèmes de combinatoire multi-objectifs, de nombreuses méthodes de résolution, exactes ou approchées, ont été proposées. Nous présentons dans la section suivante des notions essentielles et des méthodes classiques utilisées en optimisation combinatoire multi-objectifs.

1.1.3 Résolution par des méthodes exactes

Face à un problème multi-objectifs, l'une des premières idées de résolution qui nous vient à l'esprit est de le transformer en un problème mono-objectif, c'est ce qu'on appelle la scalarisation [Wierzbicki, 1986a]. Elle permet de convertir les différentes performances des solutions en une unique valeur, ce qui nous permet de comparer des solutions qui n'étaient jusqu'alors pas comparables. En effet, comme vu dans la section précédente, il est impossible de comparer certains vecteurs de performances avec la dominance de Pareto. Grâce à la scalarisation, on obtient un score qui permet d'ordonner les solutions, et d'utiliser les méthodes de résolution conçues pour le cas mono-objectif. En faisant varier les paramètres de la scalarisation, on peut

ainsi engendrer différentes solutions efficaces et obtenir l'ensemble des points non-dominés, ou une bonne approximation.

Dans cette section, on présente deux méthodes de scalarisation par agrégation des performances, suivies de la méthode appelé ε-contrainte.

Méthode de scalarisation avec la somme pondérée

Il existe de nombreuses fonctions d'agrégation dans la littérature mais la plus connue et la plus utilisée reste certainement la somme pondérée [Geoffrion, 1968].

Definition 1.1.18 (Somme pondérée)**.** *La somme pondérée assigne à chaque objectif y_i un poids λ_i de sorte que la valeur globale d'une solution x est donnée par :*

$$f_\lambda(x) = \sum_{i=1}^{p} \lambda_i y_i(x), \quad avec \quad \sum_{i=1}^{p} \lambda_i = 1 \; et \; \lambda_i \geq 0 \tag{1.1}$$

Étant donné un jeu de poids $\lambda = (\lambda_1, \ldots, \lambda_p)$, minimiser la fonction f_λ conduit à la génération d'une solution efficace. Faire varier le paramètre λ peut permettre de générer différents points non-dominés, comme illustré dans l'exemple ci-dessous.

Exemple 1.1.4. *Dans la Figure 1.5, on considère un problème bi-objectifs possédant 6 points non-dominés.*

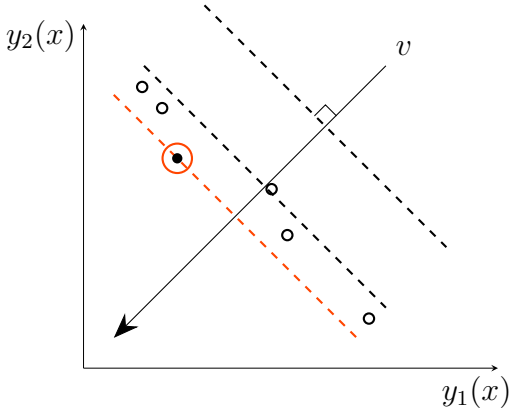


FIGURE 1.5 – Illustration de la scalarisation avec une somme pondérée.

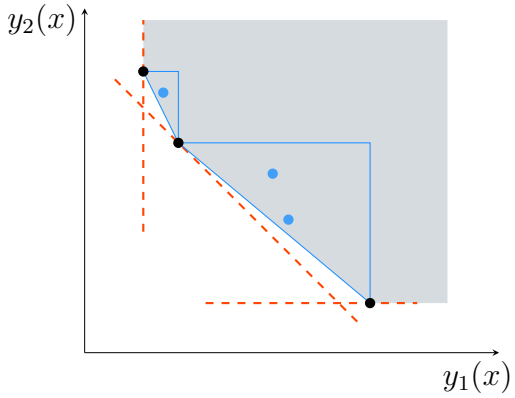


FIGURE 1.6 – Limite de la somme pondérée sur des zones non-convexes.

On applique la méthode de scalarisation par somme pondérée avec le jeu de poids $\lambda = (\lambda_1, \lambda_2) = (0.55, 0.45)$. Ainsi, le problème d'optimisation mono-objectif considéré est :

$$\min_{x \in \mathcal{X}} f_\lambda(x) = \lambda_1 y_1(x) + \lambda_2 y_2(x) = 0.55 \times y_1(x) + 0.45 \times y_2(x)$$

Une courbe d'indifférence est définie par $f_\lambda(x) = k$, pour une valeur donnée k de la fonction objectif. Elle correspond donc à l'ensemble de tous les points (réalisables ou non)

possédant la même performance globale, qui est égale à k. Sur la figure, les droites en pointillé correspondent à trois courbes d'indifférence. Minimiser la fonction objectif revient à chercher la plus petite valeur de k pour laquelle il existe une solution $x \in \mathcal{X}$ vérifiant $f_\lambda(x) = k$. Graphiquement, diminuer la valeur de k revient à déplacer la courbe d'indifférence perpendiculairement à la direction d'optimisation, qui est donnée par le gradient de la fonction objectif (λ). En minimisation, il s'agit de suivre cette direction dans le sens opposé du gradient pour diminuer la valeur de la fonction objectif. Pour l'exemple de la Figure 1.5, cela revient à suivre le sens du vecteur v. Dans cet exemple, le point encerclé en rouge est celui qui minimise la fonction objectif pour le jeu de poids donné en entrée. Il faut ensuite réitérer cette méthode avec plusieurs jeux de poids différents, afin d'essayer de trouver les autres points non dominés du problème, représentées ici par des points blancs cerclés de noir (pour insister sur le fait qu'ils n'ont pas encore été découverts). Cependant, la somme pondérée présente un désavantage important. En effet, elle ne permet pas de retourner un point non-dominé à l'intérieur de l'enveloppe convexe étendue de $\mathcal{Y}$. Cette limite de la somme pondérée est observable sur la Figure 1.6. On constate que la non-convexité du front de Pareto ne permet pas à cet opérateur linéaire de retourner les points non dominés présents dans les deux zones triangulaires bleues. Ces points ne peuvent pas être obtenus par minimisation d'une somme pondérée. Pourtant, ils sont non-dominés et de bon compromis, ils pourraient être intéressants. Les seuls points accessibles sont ceux situés sur les bords de l'enveloppe convexe, comme illustré par les traits en pointillé rouges représentant les courbes d'indifférence associées à trois sommes pondérées différentes.

Comme illustré dans l'exemple, la somme pondérée ne permet pas de retourner un point non dominé présent dans une région non convexe, ce qui est un problème quand l'objectif est de générer l'ensemble des points non dominés. On dit que la méthode de scalarisation par somme pondérée ne vérifie pas la propriété de complétude, c'est-à-dire qu'elle ne garantit pas d'atteindre toutes les solutions efficaces du problème multi-objectifs en faisant varier les valeurs des paramètres. Cependant, lorsque l'espace est convexe, la somme pondérée possède les caractéristiques suivantes :

Théorème 1.1.1. *Pour tout problème d'optimisation combinatoire multi-objectifs dont l'image de l'ensemble des solutions est convexe, on a les propriétés suivantes :*

- *Si x est une solution faiblement efficace, alors il existe un jeu de poids λ dans l'ensemble $\{\lambda' \in \mathbb{R}^p : \lambda'_i \geq 0, \forall i = 1, \ldots, p\}$ tel que x minimise la somme pondérée f_λ.*
- *Si x est une solution efficace, alors il existe un jeu de poids λ dans l'ensemble $\{\lambda' \in \mathbb{R}^p : \lambda'_i > 0, \forall i = 1, \ldots, p\}$ tel que x minimise la somme pondérée f_λ.*

Le paragraphe suivant est dédié à une autre fonction d'agrégation, plus complète que la somme pondérée puisqu'elle permet de retourner chacune des solutions associées à chaque point du front de Pareto.

Méthode de scalarisation avec Tchebychev pondéré

De la même façon que pour la somme pondérée, Tchebychev pondéré (défini dans l'article [Bowman, 1976]) assigne un poids λ_i à chaque objectif i. Et pareillement, ce jeu de

poids donnera la direction d'optimisation. Cependant, on aura besoin d'utiliser un point de référence, le point idéal, et l'objectif devient la minimisation de la norme de Tchebycheff [Steuer and Choo, 1983, Dächert et al., 2012]. Avant de décrire formellement cette méthode de scalarisation, rappelons la définition d'une norme.

Definition 1.1.19 (Norme vectorielle sur $\mathbb{R}^p$)**.** *Une norme sur* $\mathbb{R}^p$ *est une application* $||.|| : \mathbb{R}^p \to \mathbb{R}$ *qui vérifie les propriétés suivantes :*

- *Non-négativité :* $||y|| \geq 0$
- *Séparation :* $||y|| = 0 \Leftrightarrow y = 0$
- *Inégalité triangulaire :* $||y + y'|| \leq ||y|| + ||y'||$ *avec* y' *un vecteur de* $\mathbb{R}^p$
- *Homogénéité absolue :* $||\alpha y|| = |\alpha| * ||y||$ *avec* $\alpha \in \mathbb{R}$

La norme la plus utilisée sur un ensemble de dimension finie est la norme p, avec $p \geq 1$, et définie par :

$$||y||_p = \sqrt[p]{\sum_{i=1}^{p} |y_i|^p}$$

Par exemple, elle correspond, lorsque $p = 2$, à la distance euclidienne, tandis que celle utilisée dans le cadre de la scalarisation par Tchebycheff pondéré est la norme infinie :

$$||y||_\infty = \max_{i=1,\dots,p} |y_i|$$

Une représentation graphique de normes classiques est donnée en Figure 1.7, pour un vecteur y à deux dimensions.

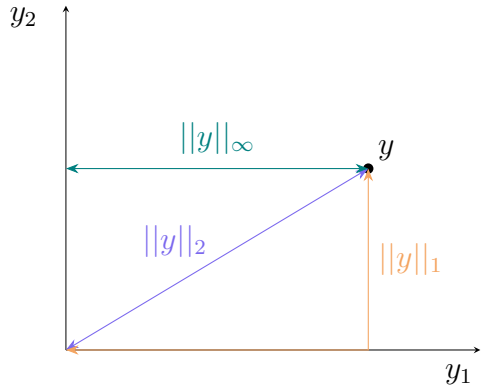


FIGURE 1.7 – Représentation graphique dans $\mathbb{R}^2$ des normes classiques.

On peut maintenant définir la méthode de Tchebycheff pondérée.

Definition 1.1.20 (Tchebycheff pondéré)**.** *La distance de Tchebycheff pondérée assigne à chaque objectif* y_i *un poids* λ_i *de sorte que la valeur globale d'une solution* x *est donnée par :*

$$f_\lambda(x) = ||y(x) - z^*||_\infty^\lambda, \quad avec \quad \sum_{i=1}^{p} \lambda_i = 1 \tag{1.2}$$

où $||y(x) - z^*||_\infty^\lambda = \max_{i=1,\dots,p} \lambda_i |y_i(x) - z_i^*|$.

En d'autres termes, $f_\lambda(x)$ correspond à la distance entre x et le point idéal au sens de la norme infinie, pondérée par le jeu de poids λ. La méthode de scalarisation par Tchebycheff pondérée permet de générer tous les points efficaces, en faisant varier le paramètre λ. Une illustration graphique de cette méthode est proposée ci-dessous, en utilisant le même exemple que précédemment.

Exemple 1.1.5. *Sur la Figure 1.8, on illustre la méthode de scalarisation avec Tchebycheff pondérée pour le poids* $\lambda = (0.45, 0.55)$.

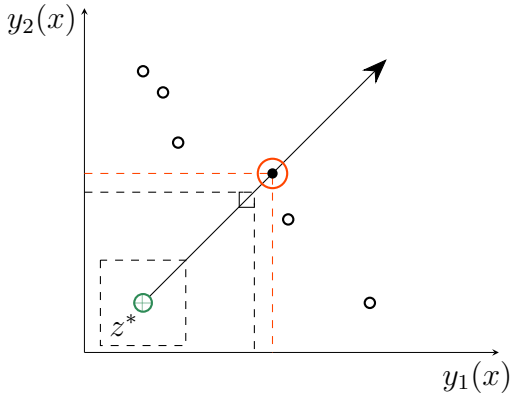


FIGURE 1.8 – Illustration de la scalarisation avec Tchebycheff.

Contrairement à la somme pondérée, une courbe d'indifférence est ici constituée de rectangles centrés en z^* *; sur le graphique sont représentées trois courbes d'indifférence en pointillé. La direction d'optimisation est donnée par le vecteur* λ *représenté par la flèche noire. Pour trouver les points minimisant la distance de Tchebycheff pondérée, il s'agit de faire grandir ce rectangle jusqu'à ce qu'il contienne un (ou des) point(s) correspondant à des solutions réalisables. Pour cet exemple, on obtient le point non-dominé encerclé de rouge.*

Remarquons que le point obtenu par la somme pondérée est différent de celui obtenu ici avec le même jeu de poids. De plus, contrairement à la somme pondérée, il est possible d'atteindre tous les points du front de Pareto en faisant varier le paramètre λ *de la distance de Tchebycheff. En effet, la forme rectangulaire des courbes d'indifférence permet d'atteindre les zones non convexes du front de Pareto, représentées par les triangles bleus de la Figure 1.6.*

Théorème 1.1.2. *Pour tout problème d'optimisation combinatoire multi-objectifs, en faisant varier* λ *dans* $\{\lambda' \in \mathbb{R}^p : \lambda'_i > 0, \forall i = 1, \dots, p\}$*, la méthode de Tchebycheff pondérée permet de générer l'ensemble des points faiblement non dominés.*

De ce fait, pour obtenir le front de Pareto, il faut ensuite éliminer les points faiblement dominés en réalisant des tests de Pareto dominance entre tous les points retournés par la méthode de Tchebycheff. Comme ces solutions peuvent être en très grand nombre, cette opération peut être coûteuse. Pour contourner ce problème, il a été proposé de modifier la fonction objectif de la manière suivante [Wierzbicki, 1986b, Wiecek and Hadavas, 1997] :

$$f_\lambda(x) = ||y(x) - z^*||_\infty^\lambda + \varepsilon||y(x)||_1$$

où ε est une constante, ce qui permet réduire les risques de génération de points en dehors du front de Pareto.

Bien que la méthode avec Tchebycheff pondérée permette de trouver l'ensemble des points non dominés (contrairement à la somme pondérée), elle présente plusieurs difficultés. Tout d'abord, on doit pouvoir calculer le point idéal de manière efficace, par exemple en résolvant p problèmes mono-objectif de la forme $min_{x\in\mathcal{X}} y_i(x)$, avec $i \in \{1, \ldots, p\}$. Ensuite, il s'agit de minimiser la fonction $f_\lambda(x = ||y(x) - z^*||_\infty^\lambda)$ qui est non-linéaire, ce qui complique la résolution. Néanmoins, il est possible d'utiliser des techniques classiques de linéarisation de l'opérateur max et de la valeur absolue $|\cdot|$.

Méthode $\varepsilon-$contrainte

Nous allons maintenant présenter la méthode appelée $\varepsilon-$contrainte [Haimes, 1971] qui est une méthode classique en optimisation combinatoire multi-objectifs. Elle diffère des méthodes vues jusqu'alors dans cette section puisqu'il n'y a pas d'agrégation des objectifs. Le principe consiste à minimiser un objectif, alors que les autres objectifs sont convertis en contraintes d'inégalité. Plus formellement, il s'agit de résoudre des problèmes de la forme :

$$\begin{cases} \min\limits_{x\in\mathcal{X}} & y_k(x) \\ \text{s.c.} & y_i(x) \leq \varepsilon_i \qquad \forall i = 1, \ldots, p, \quad i \neq k \end{cases} \tag{1.3}$$

où ε_i est une constante associée à l'objectif $y_i(x)$, pour $i \in \{1, \ldots, p\}$. Appliquons maintenant cette méthode sur le même exemple que précédemment.

Exemple 1.1.6. *Pour illustrer la méthode, on considère le front de Pareto de la Figure 1.9.*

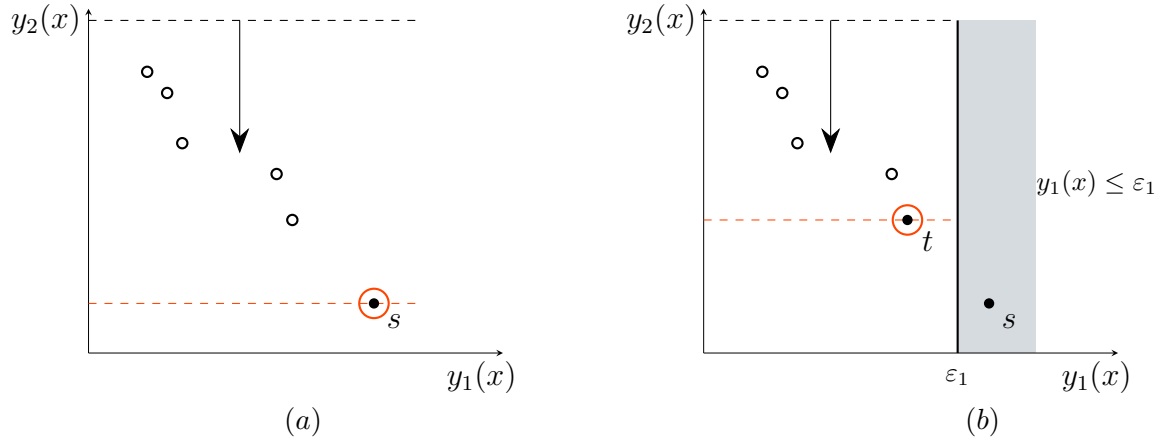


FIGURE 1.9 – Illustration de la méthode ε-contrainte sur un problème bi-objectifs.

Pour un problème bi-objectifs, il est possible de générer le front de Pareto en itérant la minimisation d'un objectif (ici y_2), avec une contrainte de type $y_1(x) \leq \varepsilon_1$ où ε_1 change

d'une itération à l'autre. Dans un premier temps, l'objectif est la recherche du point de valeur minimale sur le second objectif (cf. Figure 1.9(a)). La recherche est matérialisée par le trait en pointillé noir. Comme l'optimisation se fait seulement sur y_2, la direction d'optimisation est verticale. Le point minimisant y_2 est le dernier point réalisable rencontré par le trait en pointillé. Ce point, noté s, est ici encerclé de rouge. À partir du point s, on définit la valeur ε_1 pour l'itération suivante comme étant $y_1(s) - \xi$ où ξ est une constante positive très petite (cf. Figure 1.9(b)). Ceci dans le but d'empêcher, lors de la minimisation suivante, de retomber sur le même point. En effet, en minimisant y_2 sous la contrainte $y_1(x) \leq \varepsilon_1$, on obtient maintenant le point t. Pour obtenir le front de Pareto, on réitère cette opération jusqu'à ce qu'il n'y ait plus de solution réalisable.

Théorème 1.1.3. *Pour tout problème d'optimisation combinatoire multi-objectifs, la méthode ε-contrainte retourne une solution faiblement efficace.*

Tout comme la scalarisation avec Tchebycheff pondéré, la méthode ε-contrainte permet de déterminer tous les points du front de Pareto sans condition de convexité de l'espace des objectifs, en faisant varier ses paramètres puis en retirant les solutions générées Pareto-dominées. De plus, la méthode ε-contrainte présente l'avantage de ne pas avoir besoin de point de référence. En revanche, cette méthode modifie la structure du problème d'origine par l'ajout de contraintes, ce qui peut rendre le problème plus difficile à résoudre. Cependant, cet algorithme a été adapté à de nombreux problèmes d'optimisation combinatoire multi-objectifs [Laumanns et al., 2006, Özlen and Azizoğlu, 2009, Bérubé et al., 2009, Kirlik and Sayın, 2014, Leitner et al., 2015].

Méthode en deux phases

La méthode en deux phases est une approche générale dont l'objectif est d'explorer les zones non convexes du front de Pareto. Dans la première phase, on détermine l'ensemble des points non dominés supportés. Puis, dans la seconde phase, grâce à l'ensemble déterminé dans la première phase, on génère l'ensemble des points non dominés non supportés [Przybylski et al., 2010b]. L'algorithme original, issu de la littérature, réalise la première phase par une méthode dichotomique fondée sur la scalarisation avec somme pondérée, tandis que la seconde phase utilise une méthode énumérative utilisant les points non dominés supportés préalablement générés [Ulungu and Teghem, 1995]. Cependant, de nombreuses autres techniques peuvent être utilisées pour la seconde phase [Lee and Pulat, 1993, Sedeno-Noda and González-Martın, 2001]. Les plus efficaces actuellement sont basées sur des algorithmes de ranking. Ces méthodes permettent de découvrir les points non dominés non supportés dans l'ordre croissant de valeur obtenue par une somme pondérée [Przybylski et al., 2008, Pedersen et al., 2008, Steiner and Radzik, 2008, Raith and Ehrgott, 2009b, Raith and Ehrgott, 2009a, Jorge, 2010]. La méthode en deux phases a été originalement développée pour résoudre des problèmes bi-objectifs, cependant des adaptations ont été faites afin de pouvoir l'appliquer à des problèmes tri-objectifs [Przybylski et al., 2010a, Özpeynirci and Köksalan, 2010]. Cette méthode est illustrée sur l'exemple précédent ci-dessous.

Exemple 1.1.7. *Dans la Figure 1.10, nous illustrons la méthode en deux phases.*

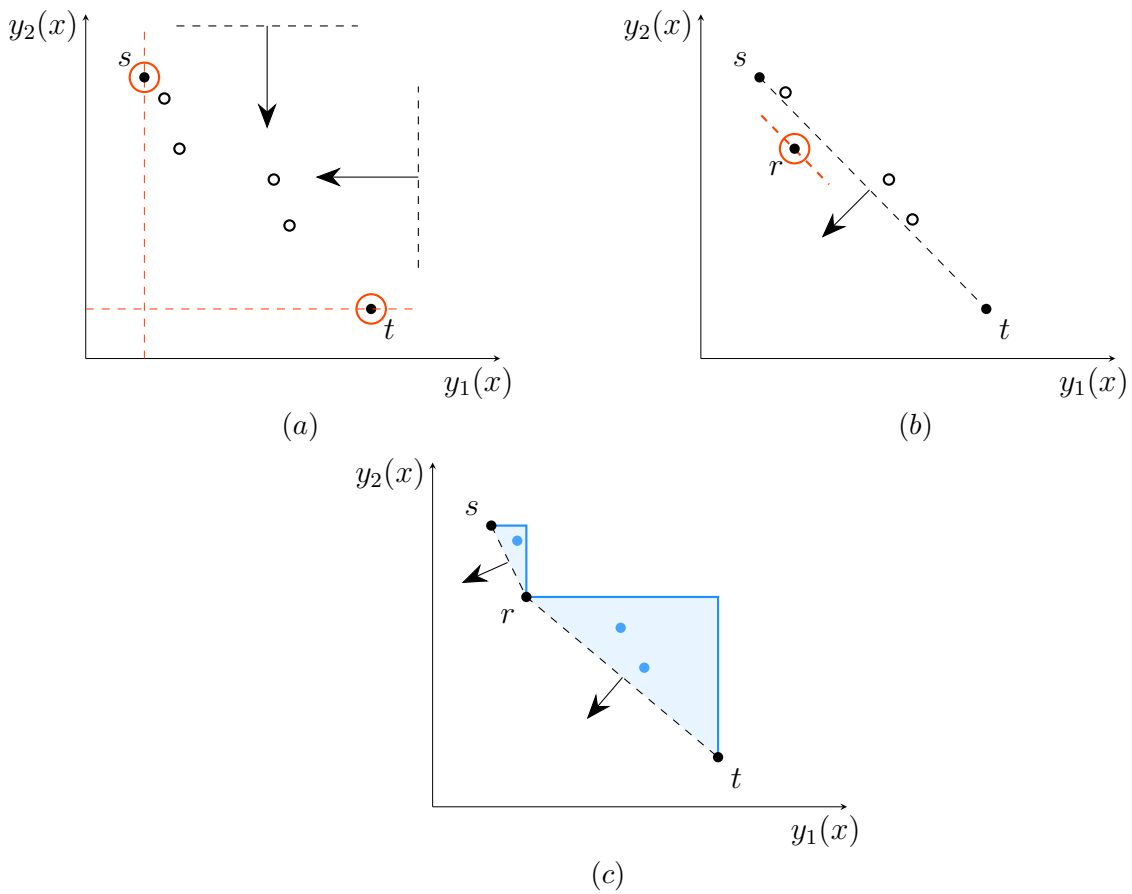


FIGURE 1.10 – Illustration de la méthode en deux phases sur un problème à deux objectifs.

La première phase, qui a l'objectif de déterminer les points non dominés supportés, est souvent réalisée en utilisant le principe de dichotomie. La dichotomie est un principe de subdivision, où l'on résout un problème en le divisant deux sous-problèmes de taille équivalente. Dans notre contexte, la phase d'initialisation consiste à trouver les points s et t minimisant respectivement le premier et le second objectif (cf. Figure 1.10(a)). Puis, on détermine un nouveau point supporté en utilisant une scalarisation avec la somme pondérée, dont le jeu de poids λ est tel que s et t appartiennent à la même courbe d'indifférence (cf. Figure 1.10(b)). On obtient ici le point r encerclé de rouge. Puis, récursivement, on détermine d'autres points non dominés supportés en résolvant les sous-problèmes entre s et r d'une part, et r et s d'autre part. Pour notre exemple, la première phase se termine ici puisqu'il n'existe aucun point non dominés supportés entre s et r, ni entre r et t.

À la fin de cette première phase, le front de Pareto n'est pas complet. En effet, il existe des points non dominés non supportés dans les triangles bleus, correspondant aux zones non convexe du front de Pareto (cf. Figure 1.10(c)) ; ce sont les points non dominés non supportés. La seconde étape consiste à trouver les points non-dominés dans chacun de ces triangles. Par

exemple, pour la zone bleue entre s et r, il faut déterminer l'ensemble des points $u = (u_1, u_2)$ non dominés vérifiant :

$$s_1 < u_1 < r_1 \quad et \quad s_2 < u_2 < r_2$$

Dans le paragraphe suivant, nous présentons un autre algorithme de résolution classique dans la littérature de l'optimisation combinatoire. Notons que celui-ci a notamment été utilisé pour réaliser la seconde phase de la méthode en deux phases [Ulungu and Teghem, 1995, Visée et al., 1998].

Branch and Bound

La méthode de branch and bound [Land and Doig, 2010, Kiziltan and Yucaoğlu, 1983] (évaluation et séparation en français) est une méthode à énumération implicite des solutions réalisables. En mono-objectif, cette méthode utilise des bornes sur la valeur optimale de sous-problèmes, permettant l'élimination de solutions partielles en détectant qu'elles ne peuvent pas conduire à des solutions efficaces. Des adaptations de cette méthode ont été proposées pour le cas bi-objectifs [Sourd and Spanjaard, 2008, Vincent et al., 2013, Przybylski and Gandibleux, 2017], dans lesquelles les bornes sont définies par un ensemble de points (au lieu d'une unique valeur). Cette méthode est basée sur trois axes principaux :

- La séparation fait référence à la méthode utilisée pour diviser intelligemment l'ensemble des solutions réalisables. Il s'agit de diviser le problème en des sous-problèmes qui ont chacun leur ensemble de solutions réalisables, tel que l'union de ces ensembles correspond à l'ensemble des solutions réalisables. De cette manière, on peut résoudre le problème initial en résolvant tous les sous-problèmes et en prenant les meilleures solutions trouvées. Ce principe est appliqué de manière récursive sur chacun des sous-ensembles de solutions obtenus, formant ainsi une structure arborescente. Par exemple, sur la Figure 1.11, le problème est divisé en considérant toutes les instanciations possibles des variables du problème, les unes après les autres.
- L'évaluation d'un sous-problème est réalisée en deux étapes : 1) le calcul de la borne inférieure et 2) l'exploitation de cette borne pour éventuellement éliminer ce sous-problème. Ces étapes sont celles qui permettent de différencier principalement le branch and bound d'une énumération classique. Il existe plusieurs procédures permettant de déterminer une borne inférieure qualitative, d'autant plus que sa qualité joue un rôle essentiel dans la réussite de cette méthode. Une façon classique de le faire est de considérer la valeur optimale de la relaxation continue du problème combinatoire considéré.
- L'élagage consiste à supprimer un sous-ensemble de solutions par comparaison de sa borne avec des solutions (complètes) déjà trouvées durant la recherche ou obtenues par des méthodes heuristiques.
- La sélection a pour objectif de choisir le prochain sous-ensemble de solutions à explorer de manière à optimiser le parcours de l'arborescence défini par la stratégie de séparation. Une stratégie efficace consiste à réaliser un parcours en profondeur, car il permet d'explorer, en priorité, les sommets les plus éloignés de la racine de l'arbo-

rescence et donc permet de trouver plus rapidement des nouvelles solutions à utiliser pour l'élagage.

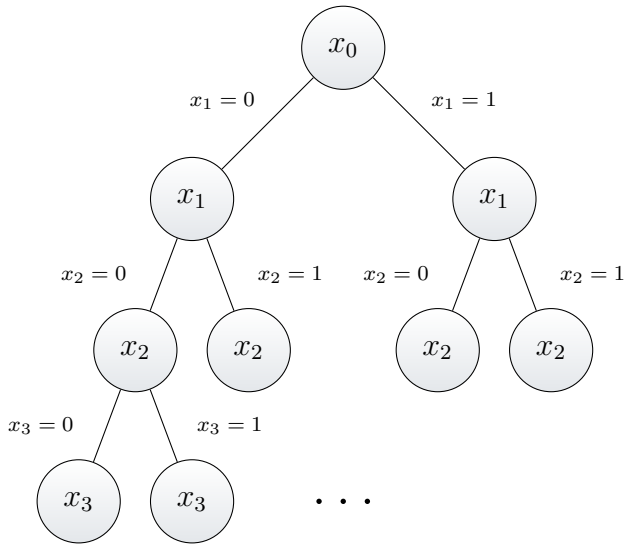


FIGURE 1.11 – Structure d'arbre d'énumération.

Dans le pire des cas, on peut tout de même être amené à faire une énumération explicite de toutes les solutions, mais en pratique cela permet d'obtenir les solutions efficaces en un temps limité.

Programmation dynamique

La programmation dynamique consiste à décomposer un problème en sous-problèmes plus petits, de calculer les solutions optimales de ces sous-problèmes et de les mémoriser, pour pouvoir construire efficacement la ou les solutions optimales du problème initial. Cette approche s'appuie sur le principe d'optimalité de Bellman [Bellman, 1966] qui énonce qu'une solution optimale pour un problème donné s'obtient en combinant les solutions optimales de ses sous-problèmes. Ce principe d'optimalité est illustré dans l'exemple ci-dessous.

Exemple 1.1.8. *On considère le problème consistant à trouver le chemin élémentaire le plus court pour aller du sommet A au sommet F, problème illustré par le graphe orienté en Figure 1.12. On rappelle qu'un chemin élémentaire est un ensemble d'arcs successifs ne passant pas deux fois par un même sommet.*

Dans ce problème, le chemin le plus court est $A - C - F$. Le sous-problème du chemin le plus court du sommet A au sommet C est de prendre l'arc $A - C$, qui fait bien partie de la solution optimale du problème initial. Remarquons que le principe de Bellman n'est pas respecté par tous les problèmes. Par exemple, en Figure 1.12, si le problème devient "trouver le chemin élémentaire le plus long allant du sommet A au sommet F", alors la solution optimale est $A - C - E - B - F$. Pour le sous-problème de trouver le chemin le plus long du sommet A au sommet C, on obtient le chemin $A - B - D - C$, alors que c'est l'arc $A - C$ qui est contenu dans la solution optimale du problème initial.

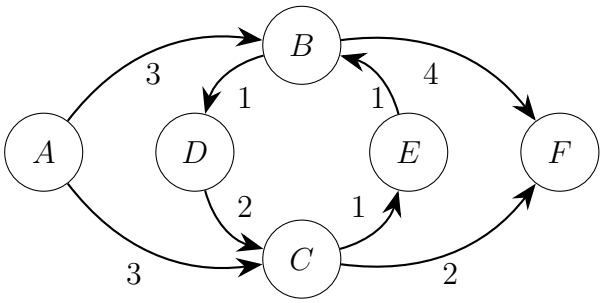


FIGURE 1.12 – Illustration du principe de Bellman.

Cette méthode a été utilisée pour résoudre des problèmes à trois objectifs [Bazgan et al., 2009], où l'utilisation de plusieurs relations de dominance permet d'écarter des solutions partielles qui ne peuvent conduire à des nouveaux vecteurs performances non dominés. Par ailleurs, de la même manière que le branch and bound, la taille des problèmes combinatoires limite l'utilisation de la programmation dynamique. Cependant, de nombreuses implémentations de cette méthode existent dans la littérature, sur des problèmes combinatoires difficiles satisfaisants le principe de Bellman [Martins, 1984, Klamroth and Wiecek, 2000, Bazgan et al., 2009].

On sait que certains problèmes d'optimisation combinatoire peuvent être résolus par des algorithmes efficaces pour leur version mono-objectif. Cependant, les problèmes combinatoires deviennent souvent plus difficiles en version multi-objectifs. S'il existe une large bibliographie au sujet des méthodes exactes de résolution d'un problème combinatoire multi-objectifs, il semble que sa résolution à l'aide d'une métaheuristique soit encore plus large. Les métaheuristiques permettent une résolution efficace mais ce sont des méthodes non exactes. Elles peuvent avoir une garantie de performance sur la qualité de la solution sous certaines hypothèses mais sont, le plus souvent, sans garantie d'optimalité mais possèdent à la place des garanties expérimentales ; la qualité de la solution trouvée est uniquement évaluée via un ensemble d'expérimentations.

1.1.4 Résolution par des métaheuristiques

Une métaheuristique est un algorithme de haut niveau d'abstraction, indépendant des problèmes (donc générique), et qui fournit un ensemble de stratégies permettant de développer des algorithmes d'optimisation heuristiques [Glover and Kochenberger, 2006, Teghem and Pirlot, 2002, Sörensen and Glover, 2013]. On distingue les heuristiques des métaheuristiques car les premières dépendent souvent du problème. En effet, une heuristique est généralement définie pour un problème précis ou exploite des informations dépendantes du problème donné. Une métaheuristique est générique et peut être adaptée à des problèmes très différents.

De nombreuses métaheuristiques puisent leurs origines dans des phénomènes naturels, allant d'une colonie de fourmis [Dorigo et al., 1996] à la métallurgie [Kirkpatrick, 1984]. Contrairement aux algorithmes de la section précédente, les métaheuristiques sont souvent

non déterministes, et quand ses choix sont aléatoires, on parle de méthode stochastique. Très souvent, les métaheuristiques n'ont pas de garantie de performance sur la qualité des solutions retournées. Cependant, elles sont très utiles en pratique car elles permettent de résoudre des problèmes dont on ne dispose pas (encore) de méthode exacte efficace.

On distingue quatre grandes familles de métaheuristiques : les méthodes basées sur une solution unique, aussi appelées "recherche locale", celles basées sur une population d'individus, les méthodes constructives et les méthodes hybrides [Glover and Sörensen, 2015]. Dans une recherche locale, l'intention générale est de guider la recherche dans des régions prometteuses ; on parle d'intensification. Les méthodes basées sur la population permettent quant à elles d'explorer différentes zones de l'espace de recherche ; on parle de diversification. Pour avoir une bonne performance avec une métaheuristique, un équilibre entre intensification et diversification est souvent primordial [Blum and Roli, 2003, Yang et al., 2014]. Les méthodes constructives permettent de former une solution intéressante à partir des éléments qui la compose ; contrairement à la recherche locale, il s'agit de construire des solutions au lieu d'explorer un voisinage de solutions. Enfin, les méthodes hybrides consistent à combiner les idées de différentes métaheuristiques.

Recherche locale

La recherche locale naît de la volonté de relier certaines solutions entre elles, puisque le caractère combinatoire de certains problèmes empêche une énumération complète. C'est une métaheuristique qui consiste à passer d'une solution réalisable à une autre en utilisant une fonction de voisinage. Une fonction de voisinage est une fonction qui, étant donné une solution, retourne un ensemble de solutions proches structurellement. Cet ensemble de solutions s'appelle le voisinage de la solution. On peut distinguer différentes familles d'heuristiques selon la manière dont est utilisée la fonction de voisinage.

Méthodes de descente. Il s'agit de passer de solutions en solutions, en sélectionnant à chaque itération une solution de meilleure qualité dans le voisinage de la solution courante. Le processus peut par exemple s'arrêter lorsqu'il n'est plus possible d'améliorer la solution courante ou encore lorsque le temps imparti est dépassé. La qualité de la solution retournée dépend de la solution initiale et de la fonction de voisinage utilisée.

Le choix de la solution suivante dans le voisinage peut se faire de différentes manières :

— Hill Climbing : On sélectionne la solution de meilleure qualité parmi le voisinage entier.

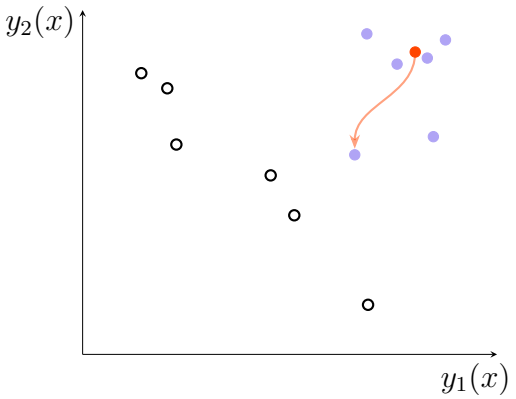
— First Improvement Hill Climbing : On construit le voisinage itérativement, et on s'arrête dès qu'on trouve une solution de meilleure qualité.

En optimisation mono-objectif, on peut aussi choisir de sélectionner une solution du voisinage de qualité égale à la solution courante, et pas nécessairement de qualité strictement supérieure. En optimisation multi-objectifs, il peut y avoir plusieurs voisins non-dominés dans un voisinage. Dans ce cas, il faut bien définir la fonction de sélection. Par exemple, dans [Basseur et al., 2012], le meilleur voisin est sélectionné sur la base du calcul d'un indicateur

de performance, plus précisément l'hypervolume de l'espace des objectifs dominé par la solution considérée. Une autre option est de sélectionner l'ensemble des voisins non-dominés pour ensuite appliquer de nouveau la fonction de voisinage sur l'ensemble de ses voisins. Cette méthode est une extension directe de la recherche locale en multi-objectifs et est couramment appelée recherche locale de Pareto [Angel et al., 2004, Basseur, 2006, Paquete et al., 2004].

Une illustration du Hill Climbing est présentée ci-dessous sur un problème bi-objectifs.

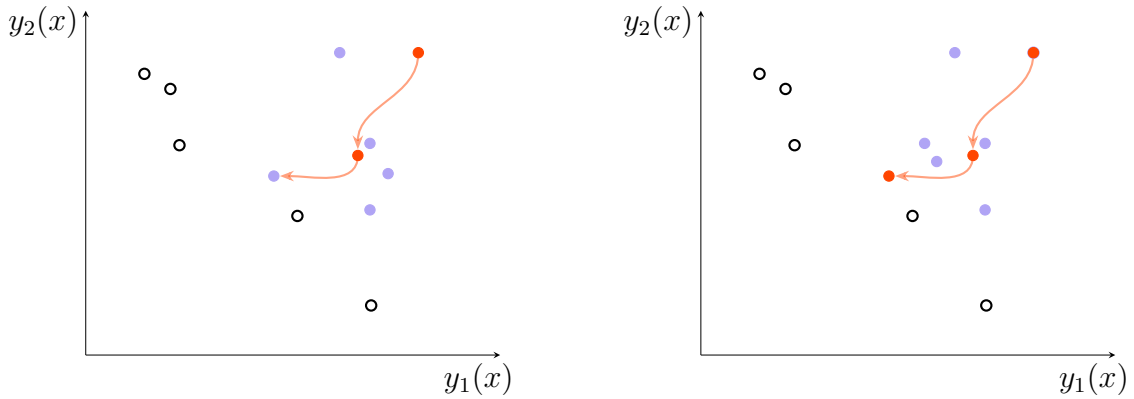
Exemple 1.1.9. *Sur la Figure 1.13, on présente une exécution possible du Hill Climbing sur un problème avec le même front de Pareto que celui de la section précédente. Sur cette figure, le front de Pareto est représenté par des cercles noirs.*



(a)

FIGURE 1.13 – Recherche locale sur deux objectifs.

La Figure 1.13(a) correspond à la première itération. La solution initiale est en rouge, et ses voisins sont représentés en mauve. On se déplace vers la meilleure solution du voisinage, c'est-à-dire celle dont l'image Pareto-domine toutes les autres.



(b) (c)

FIGURE 1.13 – Recherche locale sur deux objectifs.

Ce déplacement est représenté par la flèche rouge. Dans la Figure 1.13(b), on utilise cette solution pour générer un ensemble de solutions par le biais de fonction de voisinage. Cette fois-ci, deux voisins sont non Pareto dominés. La procédure se déplace vers une de ses deux solutions (arbitrairement) pour l'itération suivante. Sur la Figure 1.13(c), on représente la troisième itération où on obtient une solution qui n'est dominée par aucune solution de son voisinage. La procédure s'arrête donc ici, et retourne cette solution.

Les méthodes de descente ne trouvent pas nécessairement une solution efficace, et dans ce cas elles convergent vers un minimum local. Il existe d'autres algorithmes de recherche locale qui tentent de surmonter ce problème [Blot et al., 2018]. Dans le but d'éviter d'être bloqué lorsque l'on rencontre un minimum local, on peut choisir d'accepter de se déplacer vers une solution voisine qui dégrade la fonction objectif, sous certaines conditions. C'est le cas de la recherche tabou ou encore du recuit simulé.

Recherche tabou. Cette approche de recherche locale consiste à choisir à chaque itération la meilleure solution du voisinage, même quand celle-ci dégrade la solution courante, ce qui permet de sortir des minima locaux. Il existe alors un fort risque de boucle, c'est-à-dire de retomber dans le minimum local de l'étape précédente. Dans le but d'éviter cela, il a été proposé de donner une mémoire à notre algorithme [Glover, 1986]. En effet, cette méthode interdit que la solution courante soit une solution déjà explorée. Certaines de ces solutions déjà explorées sont dites taboues. Généralement ce sont les t dernières solutions visitées, ce qui impose la gestion d'une liste dite taboue de solutions explorées (de taille t) et c'est un paramètre important du problème. Sa taille, son type de gestion ou encore le mode de stockage peuvent grandement influer sur la quantité de ressources nécessaires au déroulement de cet algorithme [Gendreau, 2003].

Cet algorithme, initialement proposé en mono-objectif, a également été adapté pour résoudre différents problèmes d'optimisation combinatoire multi-objectifs [Hansen et al., 1997, Gandibleux et al., 1997, Hansen, 2000, Caballero et al., 2007].

Recuit simulé. Cette méthode de recherche locale s'inspire d'un processus métallurgique : pour faire de bons alliages (mélange de plusieurs éléments chimiques), on réalise une alternance entre des cycles chauds, où le métal est liquide, et des cycles froids, où le métal est solide [Kirkpatrick, 1984]. Le but est de minimiser l'énergie du système. L'alliage aura ainsi une structure sans défaut, alors que si le refroidissement est rapide, les atomes se figent de manière désordonnée et l'alliage possède alors des défauts (énergie élevée). Transposé à l'optimisation, on utilise une variable représentant la température T, qui définit la probabilité d'accepter de se diriger vers une solution de moins bonne qualité.

De manière plus précise, pour une solution courante x_0, on utilise la fonction de voisinage pour générer une solution voisine, notée x_1. Si cette solution est meilleure que x_0, on la sélectionne pour l'étape suivante, et on dit qu'on a fait baisser l'énergie du système. Si en revanche x_1 est moins bonne que x_0, alors on sélectionne x_1 avec une probabilité

$$e^{\frac{\triangle E}{T}} \quad \text{avec} \quad \triangle E = f(x_1) - f(x_0).$$

Dans ce cas, on dit qu'on a augmenté l'énergie. Dans le cas contraire, l'algorithme s'arrête en retournant x_0. Cette condition est appelée critère de Metropolis [Metropolis et al., 1953]. L'acceptation d'une solution dégradée permet une meilleure exploration en évitant de tomber dans un minimum local.

La température joue un rôle primordial. Ce paramètre évolue généralement pendant l'exécution de l'algorithme : on commence avec une grande valeur pour identifier des zones intéressantes de l'espace de recherche, puis on réduit sa valeur pour se concentrer sur des zones plus précises. On peut citer deux approches classiques quant à sa variation :

— Les paliers de température : garder la température constante jusqu'à atteindre un palier appelé équilibre thermodynamique, puis on diminue la température et on itère jusqu'au prochain palier.

— La décroissance continue : comme son nom l'indique, il s'agit de faire baisser la température en continu, suivant généralement la loi de décroissance $T_{t+1} = \alpha T_i$ avec $\alpha < 1$.

Cette méthode présente l'inconvénient de faire intervenir de nombreux paramètres : la valeur de la température initiale, le processus de décroissance de la température (durée du palier, valeur de α, choix de la loi de décroissance) et la condition d'arrêt (temps d'exécution). Ces paramètres ont un rôle important sur la qualité de la solution retournée.

La méthode de recuit simulé a d'abord été utilisée dans le cadre de problèmes mono-objectif, puis a été adaptée aux problèmes multi-objectifs. Une des premières adaptations vient de l'article [Ulungu et al., 1999] où une méthode de scalarisation paramétrée est utilisée pour agréger tous les objectifs. L'algorithme de recuit simulé est ensuite appliqué pour plusieurs paramètres de cette fonction de scalarisation et les solutions obtenues pour chacune des exécutions sont ensuite comparées via la dominance de Pareto, et seules les solutions non-dominées sont gardées. D'autres adaptations ont ensuite vu le jour, citons par exemple [Bandyopadhyay et al., 2008, Czyżżak and Jaszkiewicz, 1998]. Pour plus de précisions, voir l'état de l'art suivant [Suman and Kumar, 2006].

Les métaheuristiques évoquées jusqu'à maintenant sont basées sur une solution unique. Les métaheuristiques développées ensuite sont des méthodes basées sur une population d'individus.

Algorithmes évolutionnaires

Parmi les méthodes basées sur une population d'individus (par exemple des solutions), les algorithmes évolutionnaires sont les plus connus. Depuis John Holland dans les années 1960, ces algorithmes imitent le principe de l'évolution, c'est-à-dire la sélection naturelle de Charles Darwin [Bremermann, 1958, Holland, 1992]. Les algorithmes évolutionnaires sont eux-mêmes divisés en quatre catégories : les algorithmes génétiques [Goldberg et al., 1989, Fogel and Anderson, 2000], la programmation génétique [Koza, 1992], les stratégies d'évolution [Schwefel, 1981] et la programmation évolutionnaire [Fogel et al., 1966, Yao et al., 1999]. Il existe un large éventail d'implémentations très sophistiquées et très spécifiques à chaque problème. Nous allons présenter dans cette section le principe général de ces méthodes.

Les algorithmes évolutionnaires fonctionnent sur une population d'individus et utilisent

deux mécanismes pour rechercher de bons éléments : la sélection d'éléments de bonne qualité pour une fonction d'évaluation, généralement appelée fitness (fonction d'adaptation), et la combinaison de ces éléments pour créer des nouveaux à l'aide d'opérateurs spécialisés. Après combinaison, les nouveaux éléments sont insérés dans la population. Tout l'enjeu de ces algorithmes évolutionnaires est de garantir que les meilleurs éléments survivent aux différentes itérations, tout en maintenant une certaine diversité contrôlée dans la population. Pour y parvenir, les algorithmes évolutionnaires doivent faire des choix dont les principales caractéristiques sont les suivantes :

Type d'individus : classiquement, les individus choisis sont des solutions, mais ce n'est pas tout le temps le cas. Cela peut aussi être des parties de solution ou des éléments qui peuvent être transformés en solution.

Taille de la population : le nombre d'individus de la population est un autre paramètre important des algorithmes évolutionnaires, et qui est très souvent fixé tout au long de l'algorithme.

Combinaison : un nombre d'individus fixé va interagir pour former un nouvel individu ; on parle souvent de coopération pour décrire cette étape. L'analogie avec la biologie associe souvent ce nouvel individu à un enfant et ceux qui l'ont produit à ses parents. Même si combiner deux parents est le plus commun, il existe des stratégies où plusieurs individus sont utilisés, voire toute la population.

Diversification : combiner des individus entre eux peut impliquer une convergence rapide de la population vers des solutions de mauvaise qualité (par élitismes successifs). La diversification, souvent associée à la mutation dans le milieu naturel, correspond à une perturbation appliquée sur un individu. On peut aussi choisir de créer un nouvel individu en utilisant des informations collectées sur les différentes populations successives. L'idée est de produire des individus suffisamment différents de la population, pour pouvoir explorer de nouvelles zones de l'espace de recherche.

Insertion : la création de nouveaux individus par combinaison et/ou diversification peut produire des individus non conformes/réalisables. Il y a plusieurs façons différentes de réagir à cela. La première consiste à ne jamais créer de tel individu, et donc il faut utiliser des stratégies de combinaison et de diversification qui ne le permettent pas. Sinon, on peut simplement choisir d'accepter ou de refuser ce nouvel individu, avec ou sans condition.

Sélection : la sélection des individus se fait en attribuant à chaque individu une évaluation, en utilisant une fonction souvent appelée fitness. Cela peut être la fonction objectif du problème, quand les individus correspondent à des solutions. Cette étape, qui simule la pression évolutive, joue un rôle primordial dans la procédure. En effet, il faut qu'elle permette de garder les meilleurs individus tout en conservant une certaine diversité afin de ne pas converger trop rapidement vers un minimum local. Si on décide de remplacer la population de départ entièrement par des nouveaux individus, on parle de remplacement générationnel. Si c'est uniquement une partie de la population qui est remplacée alors le remplacement est dit stationnaire. Plusieurs termes sont employés au sujet de cette étape. On parle de pression sélective élevée si les chances de sélection des plus performants sont plus grandes que celles

des plus faibles. Une perte de diversité est dite élevée si la proportion d'individus non sélectionnés est grande. L'intensité de sélection s'intéresse à la valeur moyenne de la population après sélection, et peut-être utilisée comme objectif lors de la sélection pour éviter des phénomènes d'élitisme. La variance de sélection s'intéresse quant à elle à la dispersion de cette population.

Structure de voisinage : on peut choisir d'associer à un individu d'autres individus avec lesquels il peut échanger des informations.

Condition d'arrêt : tout comme les métaheuristiques précédentes, il n'existe aucun moyen de savoir si une solution trouvée correspond à une solution optimale. Cela signifie qu'un algorithme évolutionnaire ne sait jamais avec certitude quand s'arrêter, c'est une condition externe qui doit être vérifiée. Cela peut être une condition sur le temps d'exécution, le nombre d'itérations (générations), le nombre de solutions générées, ou encore l'obtention d'une solution de qualité suffisante.

Le prochain paragraphe s'intéresse à la sous-classe la plus populaire des algorithmes évolutionnaires : les algorithmes génétiques.

Algorithmes génétiques. Dans les années 1960, John Holland étudie les systèmes évolutifs et, c'est en 1975 qu'il introduit le premier modèle formel d'un algorithme génétique, le Canonical Genetic Algorithm (AGC) dans son livre "Adaptation in Natural and Artificial Systems" [Sampson, 1976]. À l'origine, il a été utilisé avec des populations composées de chaînes binaires et un remplacement générationnel. Généralement, dans un algorithme génétique, les individus formant la population sont des solutions du problème, les combinaisons sont réalisées par croisement (de vecteurs solutions), et la diversification est faite par mutation (modification d'une ou plusieurs variables). Une illustration du mécanisme général est donnée en Figure 1.14.

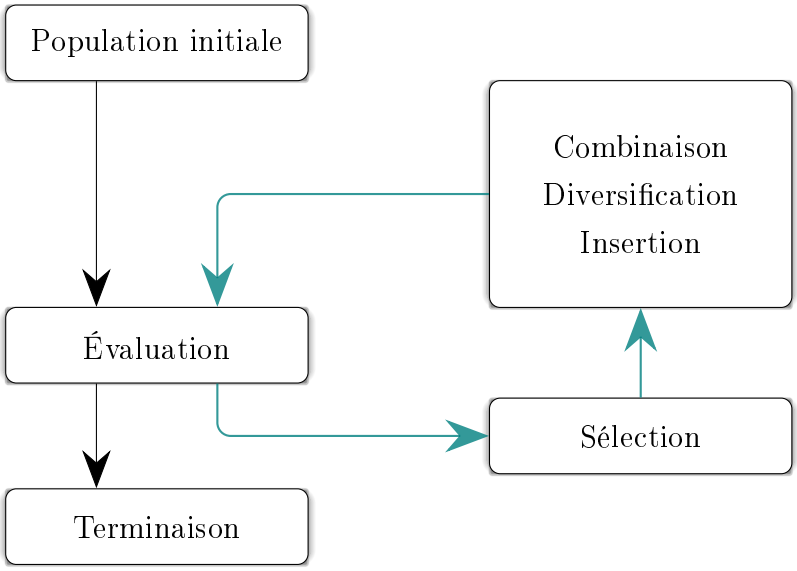


FIGURE 1.14 – Principe général des méthodes génétiques.

Parmi la multitude de méthodes utilisant des algorithmes génétiques pour la résolution de problèmes d'optimisation combinatoire multi-objectifs, on peut citer la populaire méthode NSGA développée par [Srinivas and Deb, 1994]. Une version améliorée, appelée NSGA-II, est ensuite proposée en 2002 [Deb et al., 2002]. Si NSGA n'utilise aucun principe d'élitisme, ce n'est pas le cas de NSGA-II qui utilise la sélection par tournoi, dont la pression de sélection dépend de la taille des tournois. Cet élitisme est contrebalancé par une étape de sélection qui utilise la crowding distance, qui est une mesure permettant de donner une estimation de la densité des solutions autour d'une solution de la population :

Definition 1.1.21 (Crowding distance [Deb et al., 2000])**.** *La crowding distance d'un point a s'obtient en calculant, pour chaque fonction objectif y_i, l'écart de valeurs entre deux solutions de la population : celle qui minimise la valeur de y_i parmi les solutions qui ont une valeur plus grande que a, et celle qui maximise la valeur de y_i parmi les solutions qui ont une valeur plus petite que a. La crowding distance est égale à la moyenne de ces écarts de valeurs sur les différents objectifs. Les points extrêmes, c'est-à-dire, minimisant un des objectifs, ont une crowding distance infinie.*

La crowding distance d'un point a donne donc une estimation du périmètre du cuboïde formé par les points les plus proches de a dans la population. Cette distance permet d'augmenter la diversification en favorisant les individus de la population ayant la plus grande crowding distance. La Figure 1.15 représente la crowding distance pour un point a dans une population de six points ; sa distance est ici égale à $\frac{d_1+d_2}{2}$.

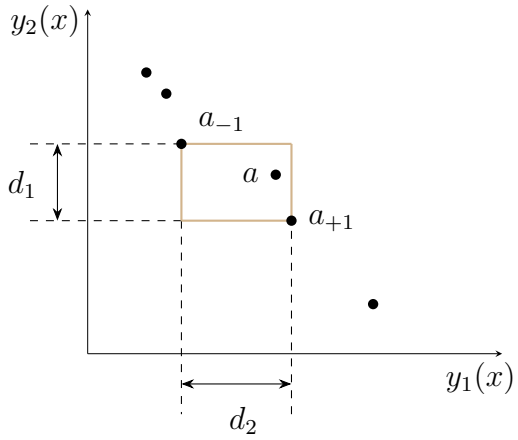


FIGURE 1.15 – Illustration de la crowding distance pour le point a.

NSGA-II est un algorithme génétique fondé sur la crowding distance, qui commence avec une population initiale aléatoire, et qui possède les particularités suivantes. L'étape de sélection commence par diviser la taille de la population par deux, pour réaliser des croisements et mutations sur les solutions les plus prometteuses. Pour cela, les individus de la population sont d'abord regroupés selon le rang de Goldberg [Grefenstette, 1993]. Le premier groupe, noté F_1, est composé des solutions associées aux points non-dominés au sens de Pareto (appelé premier front). Puis, on élimine ces solutions de la population et on détermine celles

associées aux nouveaux points non-dominés, qui constitueront le deuxième groupe, noté F_2 (et appelé deuxième front). On itère jusqu'à ce qu'il n'y ait plus de solutions dans la population. La Figure 1.16 représente les différents groupes obtenus pour une population de solutions évaluées selon deux objectifs : F_1 est formé des points encerclés de rouge, F_2 de ceux encerclés de vert et F_3 de ceux en violet.

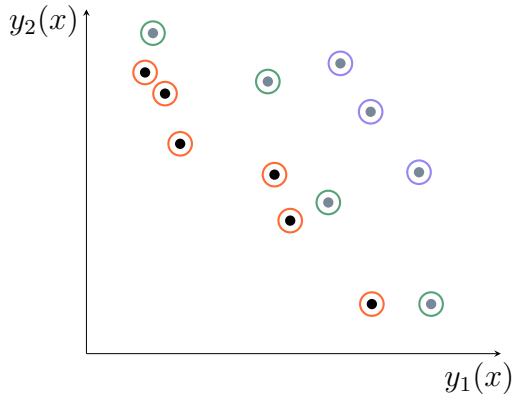


FIGURE 1.16 – Représentation de rangs dans le front de Pareto.

Les solutions sont ensuite comparées de la manière suivante : le point $a \in F_k$ est meilleur que le point $b \in F_l$, noté $a \prec b$, si et seulement si $k < l$, ou bien $k = l$ et la crowding distance de a est plus grande que celle de b. La population est alors réduite de moitié, en ne gardant que les solutions les mieux classés selon $\prec$. En pratique, pour une population de taille S, cela revient à conserver les solutions contenues dans les fronts $F_1, \ldots, F_k$ tels que $|F_1| + \ldots + |F_k| \leq \frac{S}{2}$ et $|F_1| + \ldots + |F_{k+1}| > \frac{S}{2}$, et à compléter cet ensemble de solutions par les solutions de F_{k+1} qui maximise la crowding distance, de sorte à conserver en tout $\frac{S}{2}$ solutions. Cet ensemble de solutions est ensuite utilisé pour créer de nouvelles solutions, par mutations et croisements ; ces opérateurs dépendent du problème considéré et ne sont pas liés à la description du principe général de la méthode. Ensuite, la sélection se fait par tournoi [Miller et al., 1995] selon $\prec$. Cela se déroule de la façon suivante : dans la population, on sélectionne deux individus au hasard et celui avec le meilleur classement selon $\prec$ l'emporte. On répète ce processus S fois de manière à ne garder que les S individus nécessaires à la conception de la nouvelle génération. Plus la taille du tournoi est petite et plus la pression de sélection est faible, et à l'inverse, plus on l'augmente, plus la pression est forte. L'intensité de la sélection évolue donc de la même manière en fonction de la taille du tournoi. La variance de cette méthode est quant à elle assez élevée (dispersion des individus) pour deux raisons : 1) on utilise un tournoi, et 2) la relation $\prec$ favorise les plus grandes crowding distances, ce qui permet d'obtenir une meilleure répartition des solutions dans l'espace des objectifs.

La méthode NSGA-II a été appliquée à de nombreux problèmes différents parmi lesquels nous pouvons citer, de manière non exhaustive, le problème de planification [Kannan et al., 2008], de faux diagnostic [Wang et al., 2019], de tournées de véhicules [Jozefowiez et al., 2005] ou de sélection d'éléments [Soyel et al., 2011]. Un état de l'art plus complet est disponible dans l'article [Verma et al., 2021].

Approches hybrides.

Les approches hybrides sont des approches où l'on combine plusieurs métaheuristiques. Non seulement l'hybridation améliore considérablement les performances des métaheuristiques classiques, mais surpasse souvent les autres techniques d'optimisation. Aujourd'hui, les méthodes basées sur ce principe sont les plus compétitives pour résoudre les problèmes d'optimisation [Blum et al., 2008, Prins, 2009], car les approches hybrides allient les qualités des différentes métaheuristiques utilisées. On peut citer la recherche locale évolutionnaire [Wolf and Merz, 2007] qui a pour vocation d'intensifier la recherche d'une solution de bonne qualité en couvrant les zones prometteuses adjacentes à la solution courante de manière similaire à un algorithme évolutionnaire.

Jusqu'à présent, nous avons présenté des méthodes pour trouver l'ensemble des points non dominés (ou une bonne approximation) d'un problème d'optimisation combinatoire multi-objectifs. Ces problèmes peuvent être traités de manière différente quand il s'agit de recommander une solution à une personne. Dans ce cas, il convient de déterminer la solution qui réalise le meilleur compromis selon les préférences subjectives de cette personne, ce qui fait partie des problèmes de la théorie de la décision algorithmique.

1.2 Aide à la décision multicritère

La théorie de la décision est un domaine de recherche dont l'objectif principal est d'accompagner un individu (appelé décideur) dans sa prise de décision, en tenant compte de ses préférences subjectives. Les problèmes concernés sont très variés, par exemple les problèmes de sélection [Olson, 1997], de diagnostic médical [O'connor, 2001], de gestion d'entreprise ou même de production agricole [Diak et al., 1998]. La prise de décision peut être une tâche complexe, et selon la difficulté rencontrée, on peut diviser la théorie de la décision en trois sous-domaines : la théorie de la décision multicritère, la décision collective (ou théorie du choix social) et la décision dans l'incertain. La décision multicritère [Roy, 1996] est caractérisée par la présence de plusieurs critères à prendre en compte, souvent conflictuels. La théorie du choix social concerne les problèmes dont la décision est à prendre par un groupe de personnes. L'exemple le plus connu est le choix des représentants du peuple par un système de vote [Arrow, 1950, Arrow, 2012]. La théorie de la décision dans l'incertain [Von Neumann and Morgenstern, 1947, Savage, 1972] étudie des problèmes dont on ne peut prédire avec certitude les conséquences d'une décision, ou si l'environnement est soumis à des variations inconnues. On parle de décision dans le risque quand la probabilité des différents états possibles est connue.

Historiquement, l'aide à la décision a été présenté comme le résultat d'un processus à trois étapes [Simon, 1960] :

1. Renseignement : collecte d'informations sur le problème et analyse du contexte, des acteurs, des variables, des actions, et de leurs conséquences.
2. Conception : identification des solutions possibles au problème.

3. Sélection : phase d'évaluation des solutions possibles et choix de la meilleure solution selon le critère d'évaluation considéré.

Cependant, cette description n'a pas fait l'objet d'un consensus, ce qui a conduit à la proposition d'autres définitions quelques années plus tard [Pounds, 1965, Rubenstein and Haberstroh, 1960]. Ainsi, ont été ajoutées trois autres étapes :

— Communication : incitation des acteurs, de l'assistance (apportée par un(des) analyste(s)) et du décideur à coopérer pour une meilleure compréhension du problème.
— Implémentation : élaboration algorithmique permettant de recommander la meilleure solution au décideur.
— Explicabilité : validation et explication de la solution finale, recommandée par l'assistance.

Le type de recommandation dépend de la problématique considérée [Roy, 2005, Grabisch, 2005] :

— Choix : détermination de la meilleure solution selon les préférences subjectives du décideur.
— Tri : affectation de chaque solution à une catégorie prédéfinie (par exemple "bon" ou "à rejeter"), permettant une évaluation intrinsèque des solutions du problème.
— Rangement : définition d'une relation d'ordre partiel ou total permettant le classement des solutions par ordre de préférence.
— Scorage : attribution d'un score à chaque alternative.
— Description : production d'informations sur les choix qui peuvent être effectués.

Dans le cadre de cette thèse, on ne considérera que le problème de choix.

Il existe plusieurs types d'acteurs lors d'une prise de décision, notamment le décideur et l'analyste [Giard and Roy, 1985]. Mais il est utile de préciser qu'il peut y avoir d'autres intervenants, par exemple des personnes pouvant influencer l'avis du décideur ou des individus subissant les conséquences de la décision. Ces aspects ne sont pas développés dans cette thèse, nous nous plaçons dans le cadre de problème où nous interagissons avec un décideur unique.

1.2.1 Modélisation des préférences

On considère un problème de décision multicritère où $\mathcal{X}$ est l'ensemble des solutions (alternatives) possibles, et $y : \mathcal{X} \to \mathbb{R}^p_+$ est une fonction qui associe un vecteur de performances $y(x) = (y_1(x), \ldots, y_p(x))$ à chaque solution $x \in \mathcal{X}$, donnant son évaluation sur les p différents objectifs/critères du problème. Sans perte de généralité, on suppose ici que y_i, avec $i \in \{1, \ldots, p\}$, est un critère à minimiser aux yeux du décideur. L'image des solutions réalisables dans l'espace des objectifs est notée $\mathcal{Y}$. Il s'agit pour l'analyste de déterminer la meilleure solution pour le décideur.

La dominance de Pareto permet de supprimer du problème des alternatives qui sont incontestablement mauvaises, en comparaison avec d'autres. Néanmoins, la dominance de Pareto peut laisser de nombreuses solutions incomparables, ce qui ne permet pas directement de formuler une recommandation au décideur. Pour y parvenir, on peut raffiner la dominance de Pareto en y ajoutant des informations sur les préférences du décideur. Il s'agit de définir

une relation $\precsim$ plus riche que la dominance de Pareto telle que $a \precsim b$ si et seulement si a est préférée à b. On peut distinguer sa partie stricte $\prec$ de sa partie symétrique $\sim$ définies respectivement par : $a \prec b$ si et seulement si $a \precsim b$ et non $b \precsim a$, et $a \sim b$ si et seulement si $a \precsim b$ et $b \precsim a$.

Pour définir la relation $\precsim$, une manière de procéder est d'étudier la manière dont le décideur définit la résultante des performances sur les différents critères, classiquement réalisée par agrégation ; on ramène un vecteur performance $y(x) = (y_1(x), \ldots, y_p(x)) \in \mathbb{R}^p_+$ à une unique valeur. Deux façons différentes peuvent être considérées pour agréger les vecteurs de performances de deux solutions à comparer [Perny, 2000, Grabisch, 2003] :

— "Agréger puis comparer" : ces approches agrègent tous les critères en un seul, comparable à un score et appelé critère de synthèse. Pour comparer deux solutions, on compare simplement leur score, à l'image de la théorie de l'utilité multi-attributs (MAUT) [Keeney et al., 1993, Dyer, 2005] ou de l'analyse multicritère hiérarchique (AHP) [Vaidya and Kumar, 2006], qui reposent sur ce principe.

— "Comparer puis agréger" : les méthodes de surclassement, à l'inverse, comparent critère par critère les deux alternatives avant d'agréger les comparaisons obtenues dans le but de déterminer si une alternative est meilleure qu'une autre. Parmi les méthodes les plus connues utilisant ce principe, on peut citer ELECTRE (pour ÉLimination Et Choix Traduisant la RÉalité) [Roy, 1968] et PROMETHEE (pour Preference Ranking Organisation METHod for Enrichment Evaluations) [Mareschal et al., 1984].

Remarquons que l'approche "Agréger puis comparer" nécessite que les critères soient commensurables pour que la comparaison des valeurs agrégées ait du sens, ce qui n'est pas le cas de l'approche "Comparer puis agréger".

Dans cette thèse, nous nous concentrons sur le cadre de la théorie de l'utilité multi-attributs (MAUT), où la valeur d'une solution est exprimée par une évaluation globale (un score) obtenue par une fonction d'agrégation $f_\lambda : \mathcal{X} \to \mathbb{R}^+$ paramétrée par λ (par exemple un jeu de poids). Ainsi, le décideur préfère la solution x à la solution x' si et seulement si $f_\lambda(x) \leq f_\lambda(x')$. Formellement :

$$x \precsim x' \Leftrightarrow f_\lambda(x) \leq f_\lambda(x')$$

Par exemple, lorsque les critères correspondent à des coûts comme le temps ou le prix, on peut vouloir exprimer le fait qu'un critère soit plus important qu'un autre en utilisant une somme pondérée. Néanmoins, dans la section 1.1.3, nous avons vu que la somme pondérée ne permettait pas de représenter des préférences pour des points situés à l'intérieur de l'enveloppe convexe des solutions, ce qui constitue une limite descriptive de cet opérateur. Ce constat a conduit les chercheurs en aide à la décision à étudier d'autres agrégateurs, plus sophistiqués, permettant de prendre en compte des préférences plus complexes. Il existe une large bibliographie de fonctions d'agrégation, selon le type de comportement que l'on souhaite traduire. Par exemple, si les critères correspondent aux utilités de différents agents, on peut vouloir utiliser un modèle permettant d'exprimer une certaine équité, comme l'agrégateur OWA (pour Ordered Weighted Averaging) avec un jeu de poids décroissants. Les prochains paragraphes sont dédiés à la description des fonctions d'agrégation utilisées dans ces travaux, qui sont OWA et l'intégrale de Choquet.

Ordered weighted averaging

Ordered Weighted Averaging (OWA) [Yager, 1988], appelées moyennes pondérées ordonnées en français, fait partie des familles de fonctions d'agrégation dépendants du rang les plus simples. Elles sont simplement définies comme une somme pondérée appliquée aux vecteurs de performance triés. Pareillement à la somme pondérée, OWA est paramétrée par un jeu de poids normalisé $\lambda = (\lambda_1, \ldots, \lambda_p) \in [0,1]^p$ où λ_i est le poids associé à la performance classée en $i^{ème}$ position (c'est-à-dire la $i^{ème}$ plus grande valeur). Formellement :

Definition 1.2.1 (OWA)**.** *La valeur globale d'une solution $x \in \mathcal{X}$ au sens d'un OWA est :*

$$OWA(x, \lambda) = \sum_{i=1}^{p} \lambda_i y_{(i)}(x) \quad avec \quad \sum_{i=1}^{p} \lambda_i = 1$$

où $(.)$ est une permutation telle que $y_{(1)} \geq y_{(2)} \geq \ldots \geq y_{(p)}$.

L'exemple suivant permet de comparer les calculs pour un OWA et pour une somme pondérée.

Exemple 1.2.1. *On considère un problème tri-objectifs avec trois solutions, notée s_i, dont les évaluations sont les suivantes :*

	y_1	y_2	y_3
s_1	*4*	*10*	*7*
s_2	*6*	*1*	*10*
s_3	*9*	*4*	*4*

On considère le jeu de poids $\lambda = (0.5, 0.3, 0.2)$. Avec un OWA, on obtient les évaluations globales suivantes :

- $f_\lambda(s_1) = 0.5 \times 10 + 0.3 \times 7 + 0.2 \times 4 = 7.9$
- $f_\lambda(s_2) = 0.5 \times 10 + 0.3 \times 6 + 0.2 \times 1 = 7.0$
- $f_\lambda(s_3) = 0.5 \times 9 + 0.3 \times 4 + 0.2 \times 4 = 6.5$

La solution optimale est ici s_3. En utilisant la somme pondérée et le même jeu de poids, on obtient les évaluations globales suivantes :

- $f_\lambda(s_1) = 0.5 \times 4 + 0.3 \times 10 + 0.2 \times 7 = 6.4$
- $f_\lambda(s_2) = 0.5 \times 6 + 0.3 \times 1 + 0.2 \times 10 = 5.3$
- $f_\lambda(s_3) = 0.5 \times 9 + 0.3 \times 4 + 0.2 \times 4 = 6.5$

Dans ce cas, ce n'est pas s_3 qui est optimale mais s_2. Cet exemple permet d'illustrer l'importance du choix fonction d'agrégation puisque ce n'est pas toujours la même solution qui est sera recommandée.

L'opérateur OWA est non linéaire en x avec λ fixé mais est linéaire en λ avec x fixé. Cette propriété sera notamment utilisée plus tard dans cette thèse pour pouvoir utiliser des

programmes linéaires. On remarque aussi que l'opérateur est symétrique puisque les composantes des vecteurs performances sont triées avant de calculer la valeur agrégée. Un OWA avec des poids décroissants est souvent utilisé dans la théorie du choix social afin de favoriser les solutions efficaces bien équilibrées [Korhonen et al., 1990, Goldsmith et al., 2014]. C'est aussi le cas en optimisation combinatoire multi-objectifs où nous utilisons des poids décroissants pour donner plus d'importance aux mauvaises performances. Ceci se justifie par la propriété suivante :

Proposition 1.2.1. *Soit $a = (a_1, \ldots, a_p) \in \mathbb{R}^p$ un vecteur de réels tel qu'il existe $i, j \in \mathbb{N}$ satisfaisant $a_j < a_i$ avec $j < i$. Pour chaque $\varepsilon \in (0, a_i - a_j)$, soit $a^\varepsilon = (a_1, \ldots, a_i - \varepsilon, \ldots, a_j + \varepsilon, \ldots, a_p)$ le vecteur obtenu à partir de a en effectuant un transfert de valeur ε de la position i à la position j. On a :*

$$\Big(\forall \ell \in \{1, \ldots, n-1\}, \lambda_\ell > \lambda_{\ell+1}\Big) \Rightarrow OWA(a, \lambda) > OWA(a^\varepsilon, \lambda)$$

En d'autres termes, lorsqu'on utilise des jeux de poids décroissants, une réduction de l'inégalité entre les valeurs prises par deux objectifs diminue toujours la valeur de l'OWA. Cela est dû au fait que nous accordons plus d'importance à la plus mauvaise performance, moins d'importance à la deuxième plus mauvaise performance, et ainsi de suite. Ces transferts sont connus sous le nom de transferts Pigou-Dalton en économie [Weymark, 1981]. Outre cette propriété, l'opérateur OWA est monotone, c'est-à-dire croissant par rapport à chaque composante. De ces deux propriétés, nous pouvons conclure que les solutions minimisant un opérateur OWA sont des solutions efficaces qui ne peuvent pas être améliorées en termes de transfert de Pigou-Dalton. Ainsi, la minimisation d'un opérateur OWA avec des poids décroissants permet de retourner des solutions équilibrées, tout en assurant l'efficacité des solutions.

L'utilisation de la dominance de Lorenz, définie ci-après, permet d'éliminer des solutions non optimales au sens d'un OWA. La notion de dominance de Lorenz (L-dominance) vient de l'économie où elle a été utilisé à l'origine pour mesurer les inégalités dans les distributions de revenus. Puis, au cours des dernières années, des approches basées sur la L-dominance et intégrant le concept d'équité ont été proposées dans le domaine de l'optimisation multi-objectifs. Elle permet de sélectionner des solutions efficaces qui réalisent des compromis équilibrés entre les performances.

Definition 1.2.2 (Vecteur de Lorenz)**.** *Pour tout point $a \in \mathbb{R}^p$, le vecteur de Lorenz associé à a est définie par :*

$$L(a) = (a_{(1)}, a_{(1)} + a_{(2)}, \ldots, a_{(1)} + a_{(2)} + \ldots + a_{(p)})$$

où $(.)$ est une permutation telle que $a_{(1)} \geq a_{(2)} \geq \ldots \geq a_{(p)}$.

On note $L_i(a)$ la $i^{\text{ème}}$ composante de $L(a)$. On peut alors définir la dominance de Lorenz généralisée en utilisant la dominance de Pareto de la manière suivante.

Definition 1.2.3 (Dominance de Lorenz généralisée)**.** *La dominance de Lorenz généralisée, notée $\prec_L$, est un ordre partiel sur $\mathbb{R}^p$ définie par :*

$$\forall a, b \in \mathbb{R}^p, a \prec_L b \iff L(a) \prec_P L(b)$$

Si $a \prec_L b$, on dit que a Lorenz-domine b et que b est Lorenz-dominé par a.

En d'autres termes, pour savoir si un vecteur a Lorenz-domine un vecteur b, il suffit de calculer les vecteurs de Lorenz associés à a et b et d'appliquer ensuite la dominance de Pareto sur ces deux nouveaux vecteurs. Par exemple, pour savoir si le vecteur $a = (7, 8, 7)$ Lorenz-domine le vecteur $b = (7, 9, 6)$, on calcule $L(a) = (8, 15, 22)$ et $L(b) = (9, 16, 22)$, et on voit ici que $L(a)$ Pareto-domine $L(b)$ et donc que a Lorenz-domine b.

En remarquant que $a_{(i)} = L_i(a) - L_{i-1}(a)$ pour $i > 1$ on peut réécrire l'opérateur OWA de la manière suivante :

Proposition 1.2.2.

$$OWA(a, \lambda) = \sum_{i=1}^{p} (\lambda_i - \lambda_{i+1}) L_i(a) \qquad avec\ \lambda_{p+1} = 0$$

Ainsi un OWA à poids décroissants est une combinaison linéaire à coefficients strictement positifs des composantes du vecteur de Lorenz. Par conséquent, la relation de préférence stricte $\prec_{OWA}$, définie par $a \prec_{OWA} b$ si et seulement si $OWA(a) < OWA(b)$, correspond à un ordre partiel qui étend la dominance de Lorenz. Autrement dit, on a :

$$a \prec_L b \Longrightarrow a \prec_{OWA} b$$

Une illustration de cette dominance est présentée ci-dessous sur un problème bi-objectifs :

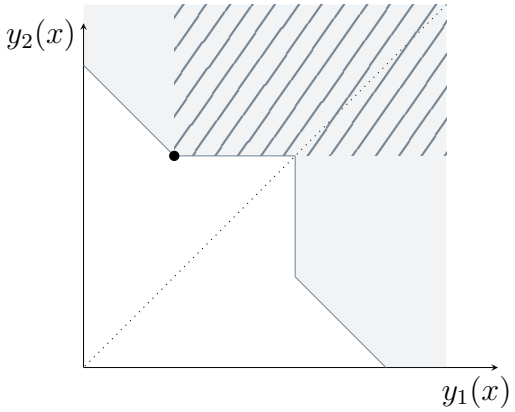


FIGURE 1.17 – Illustration de la Lorenz-dominance et Pareto-dominance.

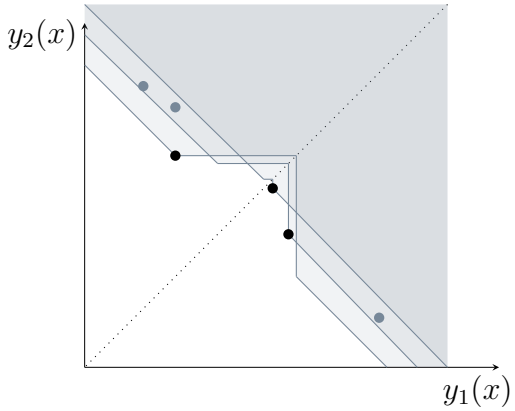


FIGURE 1.18 – Front de Pareto versus points non Lorenz-dominés.

Exemple 1.2.2. *La Figure 1.17 représente en gris le cône de dominance de Lorenz d'un point donné. Sur cette même figure est représenté en hachuré son cône de dominance de Pareto. On remarque que la dominance de Lorenz permet de couvrir un plus grand espace que la dominance de Pareto. La Figure 1.18 représente un front de Pareto composé de six points. Parmi ces six points, on peut voir que seuls trois d'entre eux sont non-dominés au sens de la dominance de Lorenz, comme le montre les trois cônes de dominance de Lorenz.*

Mais les opérateurs somme pondérée et OWA ont une limite : ils ne permettent pas de prendre en considération les dépendances entre les critères. Si on reprend l'exemple d'un trajet en voiture, la satisfaction d'avoir un trajet de courte durée et peu onéreux peut être plus élevée que la somme des satisfactions de ces deux critères pris séparément. En effet, il peut y avoir un gain supplémentaire dû à un phénomène de rareté (aller vite signifie souvent payer des autoroutes). La dépendance n'est pas nécessairement positive, elle peut aussi être négative. Par exemple, avoir un GPS et une carte routière est intéressant pour trouver son chemin, mais avoir les deux est redondant et prend de la place. La prochaine fonction d'agrégation présentée permet de modéliser ce type d'interaction entre les critères.

Intégrale de Choquet

Les intégrales de Choquet [Grabisch and Labreuche, 2010, Choquet, 1953, Schmeidler, 1986] forment une famille plus générale d'agrégateurs qui est très intéressante pour la modélisation des préférences car elle permet de modéliser différents types d'interactions entre les critères. Plus précisément, ses paramètres sont sous la forme d'une capacité (normalisée) $v : 2^N \to [0,1]$, avec $N = \{1, \ldots, p\}$, qui attribue des poids à chaque coalition de critères. Ceci permet de modéliser des interactions positives et/ou négatives entre les objectifs, couvrant une gamme importante de comportements décisionnels possibles. Formellement, une fonction de capacité est définie par :

Definition 1.2.4 (Fonction de capacité)**.** *La fonction $v : 2^N \to [0,1]$ est une capacité (normalisée) si et seulement si :*

- $v(\emptyset) = 0$, $v(N) = 1$ *(normalisation),*
- $v(A) \leq v(B), \quad \forall A \subset B \subseteq N$ *(monotonie).*

Une capacité v est dite :

- convexe (super-modulaire) quand $v(A \cup B) + v(A \cap B) \geq v(A) + v(B)$, pour tout $A, B \subseteq N$,
- additive lorsque $v(A \cup B) + v(A \cap B) = v(A) + v(B)$, pour tout $A, B \subseteq N$,
- concave (sous-modulaire) quand $v(A \cup B) + v(A \cap B) \leq v(A) + v(B)$, pour tout $A, B \subseteq N$.

Nous verrons que ces propriétés vont permettre de modéliser des comportements différents vis-a-vis de l'équité entre les critères. Nous utilisons maintenant la notion de capacité pour définir une intégrale de Choquet.

Definition 1.2.5 (Intégrale de Choquet). *La valeur globale d'une solution $x \in \mathcal{X}$ au sens d'une intégrale de Choquet est :*

$$Choquet(x, v) = \sum_{i=1}^{p} \Big(y_{[i]}(x) - y_{[i-1]}(x) \Big) v(X_{[i]}) \quad avec\ y_{[0]}(x) = 0$$

où $[.]$ est une permutation telle que $y_{[1]}(x) \leq y_{[2]}(x) \leq \ldots \leq y_{[p]}(x)$ et $X_{[i]} = \{[i], \ldots, [p]\}$ est l'ensemble des critères j tels que $y_j(x) \geq y_{[i]}(x)$.

Comme OWA, $Choquet(x, v)$ n'est pas linéaire en x pour une capacité v fixée mais est linéaire en v pour une solution x fixée. À titre d'illustration, considérons l'exemple suivant.

Exemple 1.2.3. *Soit la fonction de capacité suivante :*

	$\emptyset$	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1, 2, 3}
v	0	0.2	0.1	0.3	0.4	0.7	0.6	1

Pour une solution s_1 de vecteur performance $y(s_1) = (3, 2, 5)$, la valeur de l'intégrale de Choquet est $Choquet(s_1, v) = (2-0) \times v(\{1,2,3\}) + (3-2) \times v(\{1,3\}) + (5-3) \times v(\{3\}) = 2 \times 1 + 1 \times 0.7 + 2 \times 0.3 = 3.3$. Pour une solution s_2 avec les performances $y(s_2) = (1, 4, 3)$, la valeur de l'intégrale de Choquet est $Choquet(s_2, v) = (1-0) \times v(\{1,2,3\}) + (3-1) \times v(\{2,3\}) + (4-3) \times v(\{2\}) = 1 \times 1 + 2 \times 0.6 + 1 \times 0.1 = 2.3$. On a $Choquet(s_2, v) < Choquet(s_1, v)$, ce qui signifie que la solution s_2 est ici strictement meilleure que la solution s_1.

Dans la définition de l'intégrale de Choquet, l'utilisation d'une capacité v au lieu d'une fonction d'ensemble arbitraire garantie la compatibilité avec la dominance de Pareto en raison de la monotonie de v par rapport à l'inclusion. En d'autres termes, elle garantit que l'inégalité $Choquet(x, v) \leq Choquet(x', v)$ est satisfaite dès lors que $y(x) \prec_P y(x')$.

La famille des intégrales de Choquet comprend de nombreux agrégateurs comme cas particuliers. Par exemple, elle se résume à la famille des sommes pondérées lorsqu'on considère des capacités additives et elle correspond à la famille des agrégateurs OWA lorsqu'on utilise des capacités symétriques, c'est-à-dire telles qu'il existe une fonction non décroissante ψ satisfaisant $v(A) = \psi(|A|) \quad \forall A \subseteq N$.

Les intégrales de Choquet sont convexes lorsque v est concave (sous modulaire) et concaves lorsque v est convexe (super-modulaire) [Lovász, 1983]. Dans les problèmes de minimisation, l'utilisation d'une capacité concave permet de modéliser les préférences pour des solutions équilibrées, comme le montre la proposition suivante [Chateauneuf et al., 1999] :

Proposition 1.2.3. *Soit v une capacité concave : Pour tout $x^1, \ldots, x^q \in \mathbb{R}^p$ et tout $\mu_1, \ldots, \mu_q \in [0, 1]$ tels que $\sum_{i=1}^{q} \mu_i = 1$, on a :*

$$\Big(Choquet(x^1, v) = \ldots = Choquet(x^q, v) \Big) \Rightarrow \forall k \in \{1, \ldots, q\}, Choquet(x^k, v) \geq Choquet(\bar{x}, v)$$

où $\bar{x}$ est une solution dont le vecteur coût correspond à la moyenne des coûts des solutions $x^1, \ldots, x^q$, c'est-à-dire $y(\bar{x}) = \sum_{j=1}^{p} \mu_j y(x^j)$.

Par exemple, avec une capacité concave, si le décideur est indifférent entre les vecteurs de coût $(0, 1)$ et $(1, 0)$, alors on sait qu'il préfère le vecteur de coût $(0.5, 0.5)$ à n'importe lequel des deux autres vecteurs, puisque $(0.5, 0.5)$ est obtenu en faisant la moyenne de deux autres vecteurs ($\mu_1 = 0.5$, $\mu_2 = 0.5$).

Afin de présenter une autre formulation utile de l'intégrale de Choquet, nous fournissons maintenant une formulation alternative des capacités en utilisant les masses de Möbius :

Definition 1.2.6 (Inverse de Möbius et masses de Möbius)**.** *Toute capacité $v : 2^N \to \mathbb{R}$ est associée à une fonction d'ensemble $m : 2^N \to \mathbb{R}$ appelée inverse de Möbius, définie par :*

$$\forall A \subseteq N, m(A) = \sum_{B \subseteq A} (-1)^{|A \setminus B|} v(B),$$

de sorte que v puisse être reconstruite à partir de son inverse de Möbius comme suit :

$$\forall A \subseteq N, v(A) = \sum_{B \subseteq A} m(B).$$

Les coefficients $m(A)$, avec $A \subseteq N$, sont appelés masses de Möbius.

En utilisant la notion d'inverse de Möbius, nous obtenons la formulation suivante de l'intégrale de Choquet [Chateauneuf and Jaffray, 1989] :

$$Choquet(x, v) = \sum_{A \subseteq N} m(A) \min_{j \in A} y_j(x)$$

Ceci offre une autre interprétation de l'intégrale de Choquet : c'est une somme pondérée appliquée au vecteur de taille 2^p dont les composants sont $\min_{i \in A} y_i(x)$ pour $A \subseteq N$.

Toute capacité dont les masses de Möbius sont non négatives est appelée fonction de croyance, et est par définition monotone et d'ordre infini [Dempster, 2008, Shafer, 1976]. En utilisant l'inverse de Möbius, nous pouvons définir la notion de capacité k-additivité [Grabisch et al., 2009]. Une capacité est dite k-additive lorsque ses masses de Möbius $m(A)$ sont égales à zéro pour tout $A \subseteq N$ tel que $|A| > k$, et qu'il existe au moins un ensemble A de taille k tel que $m(A) \neq 0$. Plus formellement :

Definition 1.2.7 (Capacité k-additive)**.** *Une capacité est dite k-additive lorsque m, l'inverse de Möbius associé, vérifie les propriétés suivantes :*

$$\Big(\forall A \subseteq N, |A| > k \Rightarrow m(A) = 0\Big) \text{ et } \Big(\exists A \subseteq N, |A| = k \text{ et } m(A) \neq 0\Big)$$

Les capacités k-additives correspondent aux capacités additives lorsque $k = 1$. Pour de petites valeurs de k (supérieures à 1), les capacités k-additives sont très utiles en pratique car elles permettent de modéliser des interactions entre critères avec un nombre réduit de paramètres. Par exemple, pour la sous-classe des capacités 2-additives, les capacités sont entièrement caractérisées par $\frac{(p^2+p)}{2}$ valeurs, soit une masse de Möbius pour chaque singleton et chaque paire de critères (contre 2^p sinon).

Dans cette sous-section, nous avons vu que différentes fonctions d'agrégation pouvaient être utilisées pour représenter les préférences du décideur. Le choix de la famille de fonctions dépend de la complexité des préférences à modéliser. Néanmoins, il faut parfois faire un compromis entre la flexibilité du modèle et la complexité de l'apprentissage du paramètre permettant de représenter au mieux les préférences du décideur. En effet, bien que l'intégrale de Choquet soit plus flexible qu'une somme pondérée, elle nécessite l'apprentissage de 2^p valeurs (capacité), contre seulement p poids pour la somme pondérée.

Le point commun entre tous ces modèles, c'est qu'ils sont paramétrables. C'est ce paramètre qui nous permet d'adapter le modèle aux préférences du décideur. En effet, le score des solutions dépend du jeu de poids ou de la capacité utilisée. Pour pouvoir formuler une recommandation, notre but premier est de trouver les valeurs du paramètre permettant de modéliser au mieux les préférences du décideur. On s'intéresse donc maintenant à l'apprentissage de ce paramètre.

1.2.2 Méthodes de résolution a priori, a posteriori et interactives

Dans cette sous-section on présente les trois grandes approches pour résoudre un problème d'optimisation multi-objectifs, lorsque le but final est d'identifier la solution de meilleur compromis pour le décideur (par exemple, celle qui minimise une fonction d'agrégation).

Approches a priori

L'idée générale des approches a priori [Huédé et al., 2006] est de collecter des informations sur les préférences du décideur pour pouvoir ensuite rechercher la meilleure solution selon ses préférences. Quand ses préférences sont représentables par une fonction d'agrégation paramétrée f_λ, il s'agit de poser des questions au décideur pour déterminer les valeurs de λ correspondant au mieux à ses préférences, puis de chercher la solution optimale au sens de la fonction f_λ apprise.

L'apprentissage des meilleurs paramètres possibles peut aussi se faire via un historique ou une base de données disponible, pour les problèmes de décision qui se présentent de manière récurrente pour le décideur. Se posent alors certaines difficultés :

— Les fonctions d'agrégations sont très sensibles à leurs paramètres, c'est-à-dire que si on modifie très légèrement une de ses valeurs, on peut se retrouver avec une solution optimale complètement différente. C'est un problème si les bases de données ne sont pas très précises ou incomplète.

— Les paramètres sont très difficiles à déterminer avec précision. En effet, si l'on souhaitait déterminer le jeu de poids du décideur en lui posant des questions, cela nécessiterait d'utiliser un questionnaire beaucoup trop long pour le décideur et contenant des questions assez fines (comparaison de solutions très similaires).

— Comme la collecte des préférences a lieu avant la résolution du problème, il peut arriver que le décideur ne connaisse pas les possibilités et les limites du problème [Simon, 1960]. À ce titre, il pourrait avoir des aspirations très différentes de ce qu'il souhaite réellement et de ce qui est possible.

— Il faut ajouter que le décideur n'a pas forcément de vision claire de ses préférences [Simon, 1960]. Sa vision peut aussi évoluer avec le temps, l'information qu'il reçoit ou les solutions qu'il découvre.

Approches a posteriori

Les approches a posteriori consistent à d'abord résoudre le problème d'optimisation multi-objectifs, en calculant le front de Pareto ou une bonne représentation, puis à présenter ces solutions au décideur pour qu'il fasse un choix. La collecte des préférences a donc lieu après la résolution, à l'inverse de l'approche a priori.

L'avantage de cette approche est que l'on dispose de toutes les solutions efficaces (ou une bonne représentation), ce qui permet de ne pas relancer la résolution si les préférences du décideur changent. Néanmoins, elle présente un inconvénient majeur pour la résolution de problèmes d'optimisation combinatoire multi-objectifs, car le nombre de solutions efficaces peut être très grand. C'est ce que l'on peut observer à notre échelle dans les problèmes du quotidien : choisir simplement entre quelques dizaines d'options peut déjà s'avérer difficile, par exemple quand il s'agit de choisir une voiture. De ce fait, présenter plusieurs milliers de solutions efficaces au décideur pour qu'il fasse un choix ne semble pas être une option envisageable. Enfin, un autre inconvénient de cette approche, est qu'elle nécessite de générer le front de Pareto, ce qui peut être extrêmement coûteux voire impossible à ce jour pour certains problèmes.

Approches interactives

Dans les approches interactives (voir par exemple [Benayoun et al., 1971, Steuer, 1986, Vanderpooten and Vincke, 1989]), le décideur est sollicité pendant la recherche de sa solution préférée. Le principe général est le suivant : à chaque itération, on présente une solution efficace au décideur, le décideur exprime comment il aimerait que cette solution soit améliorée, puis on génère une solution en tenant compte de ces informations. Le processus s'arrête quand le décideur est satisfait de la solution qui lui a été présentée.

Un inconvénient de cette approche est qu'elle nécessite une implication et une certaine disponibilité du décideur. En effet, le temps entre ces interactions et leur durée a une importance ; par exemple si le décideur doit attendre une heure entre chaque proposition, cette approche semble difficile à mettre en œuvre. Par contre, elle présente au moins deux avantages. Le premier est de construire la solution avec le décideur, ce qui permet de réduire les interactions à ce qui est nécessaire pour identifier une solution satisfaisante pour le décideur. Le deuxième est que les questions sont généralement plus faciles que celles utilisées dans les méthodes a priori (qui ont besoin d'être précises) et des méthodes a posteriori qui demandent au décideur de considérer toutes les solutions efficaces (réalisant potentiellement des compromis très différents).

Dans le cadre de cette thèse, nous nous intéressons à une approche interactive particulière, appelée communément "élicitation incrémentale" dans le domaine de la théorie de la décision.

1.3 Élicitation incrémentale des préférences

Dans cette section, nous nous intéressons à l'élicitation incrémentale des préférences, qui est une approche dont le principe général est de réduire progressivement l'imprécision autour des paramètres du modèle décisionnel (fonction d'agrégation) par interaction avec le décideur. Au lieu d'apprendre précisément le paramètre permettant de représenter exactement les préférences du décideur, l'objectif est de lui poser des questions sur ses préférences, de sorte à identifier et éliminer des solutions non pertinentes pour lui, jusqu'à être en mesure de trouver sa solution préférée. L'intérêt principal de cette approche est de réduire le nombre de questions posées au décideur, puisqu'il ne s'agit pas d'apprendre le modèle de manière précise, mais plutôt de réduire suffisamment l'imprécision autour de son paramètre pour pouvoir prendre une décision.

On illustre ce principe général sur l'exemple ci-dessous.

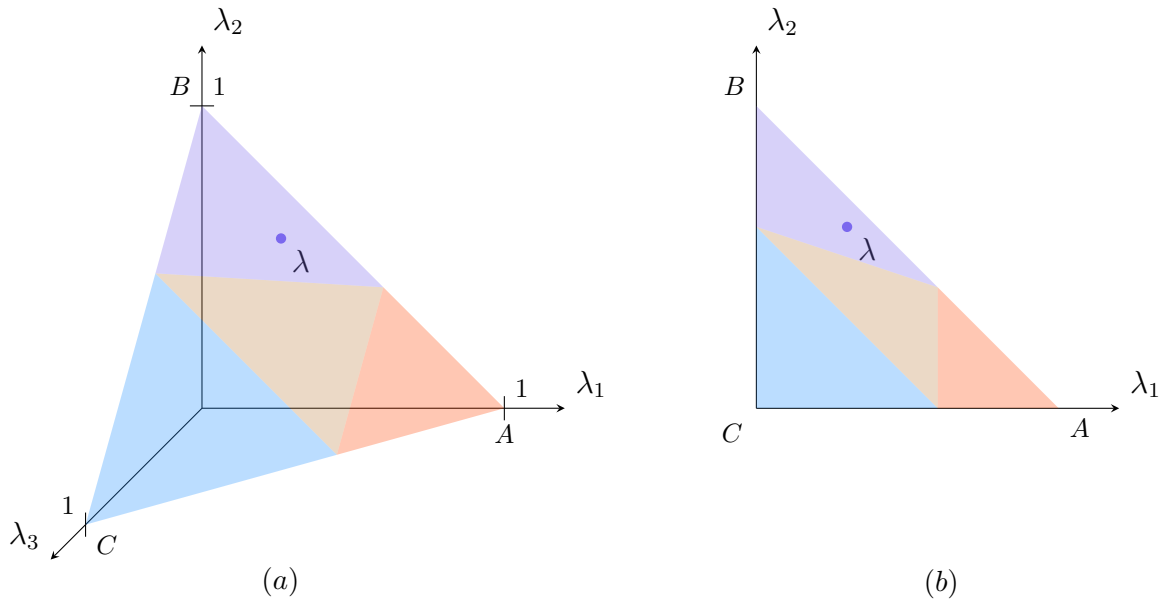


FIGURE 1.19 – Illustration de zones d'optimalités.

Exemple 1.3.1. *On considère un problème de décision tri-critères où les préférences du décideur sont représentables par une somme pondérée f_λ dont le jeu de poids est $\lambda = (0.3, 0.6, 0.1)$. Ce jeu de poids est inconnu initialement de la procédure de recommandation. À la place, il s'agit de considérer tous les jeux de poids admissibles. Sur la Figure 1.19(a), on représente l'ensemble des paramètres admissibles dans l'espace à trois dimensions (un axe par critère). Pour la somme pondérée, les jeux de poids admissibles sont définis par les contraintes suivantes : $0 \leq \lambda_i \leq 1$ pour tout $i \in \{1, \ldots, p\}$, et $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Cela correspond au triangle ABC sur la figure. À cause de cette dernière contrainte, l'ensemble des jeux de poids admissibles est contenu dans un plan. On peut aussi choisir de représenter cet ensemble dans le plan, comme c'est le cas en Figure 1.19(b). En effet, avec la normalisation, le troisième*

poids (λ_3) peut être déduit des deux autres à l'aide de la formule suivante : $\lambda_3 = 1 - \lambda_2 - \lambda_1$. Sur ces deux figures, le jeu de poids (inconnu) du décideur est représenté par un point.

Dans ce problème, il s'agit de départager quatre solutions efficaces s_1, s_2, s_3 et s_4. Pour chaque solution s_j, on représente sur la Figure 1.19 sa zone d'optimalité, qui est définie par l'ensemble des jeux de poids λ tel que $f_\lambda(s_j) \leq f_\lambda(s_k)$ pour tout $k \in \{1, 2, 3, 4\}$. Les quatre polygones colorées représentent les zones d'optimalité respectives des solutions s_1, s_2, s_3 et s_4. On voit ici que la meilleure solution est celle dont la zone d'optimalité correspond au triangle violet, puisque le jeu de poids (inconnu) du décideur est situé dans ce triangle. Ainsi, pour identifier la meilleure solution pour le décideur, il n'est pas nécessaire de connaître précisément son jeu de poids, il suffit de lui poser des questions permettant de savoir dans quelle zone se situe son jeu de poids.

Notons que l'espace des paramètres admissibles et la forme des zones d'optimalité dépendent de la fonction d'agrégation utilisée. Par exemple, avec un OWA à poids décroissants, on aurait des contraintes supplémentaires sur les paramètres, réduisant dès lors l'espace des poids admissibles sans information supplémentaire.

1.3.1 Méthodes classiques

Dans cette section, on présente deux méthodes standards en élicitation incrémentale des préférences : ISMAUT et NEMO-II.

ISMAUT

La méthode ISMAUT (pour Imprecisely Specified Multiattribute Utility Theory) est une des premières méthodes proposées rentrant dans la catégorie des méthodes d'élicitation incrémentales. Cette méthode itérative a été proposée dans le cadre de l'élicitation d'une fonction d'utilité additive (à maximiser). À chaque itération, on commence par calculer l'ensemble des solutions non dominées au sens de la relation $\succsim_U$ définie par : $a \succsim_U b$ si et seulement si $u(a) \geq u(b)$ pour tout $u \in U$ où U est l'ensemble des fonctions d'utilités additives u compatibles avec les données de préférences disponibles (par exemple, les réponses aux questions posées jusqu'ici). Si l'ensemble des non dominés au sens de la relation $\succsim_U$ est trop grand pour que le décideur puisse prendre une décision, on pose une question au décideur pour réduire U et indirectement l'ensemble des non dominées. On itère jusqu'à ce que le décideur parvienne à identifier la solution qu'il préfère dans cet ensemble. Initialement, cette méthode ne définit pas de stratégie pour le choix de la question à poser à une itération donnée, mais différentes stratégies ont été envisagées par la suite pour identifier des questions informatives permettant de réduire le nombre de questions à poser au décideur. Par ailleurs, elle a été proposée dans le cadre de problèmes sur domaine non combinatoire, ce qui permet de calculer efficacement l'ensemble des non dominées à chaque itération.

NEMO

NEMO (pour Necessary-preference-enhanced Evolutionary Multi-objective Optimizer) [Branke et al., 2010] est un algorithme évolutionnaire interactif qui a été proposé pour ré-

soudre des problèmes d'optimisation combinatoire multi-objectifs. NEMO est en fait une adaptation interactive de l'algorithme NSGA-II, qui a été présenté dans la section 1.1.4. Contrairement à NSGA-II, qui a pour objectif de déterminer une bonne approximation du front de Pareto, NEMO souhaite identifier la solution préférée du décideur au sens d'une fonction additive f_λ, en élicitant le paramètre λ en cours de résolution ; à certaines itérations, on pose une question au décideur sur ses préférences. Plus précisément, NEMO procède comme NSGA-II, en divisant la population en fronts $F_1, \ldots, F_k$, et en comparant les points au sein d'un même front avec une distance, mais NEMO se distingue de NSGA-II de la manière suivante :

— Dans NSGA-II, les fronts successifs $F_1, \ldots, F_k$ sont composés des points non dominés au sens de la dominance de Pareto. Dans NEMO, ils correspondent aux non dominés au sens de la préférence nécessaire $\prec^N$ définie par : a est meilleur que b, noté $a \prec^N b$, si et seulement si $f_\lambda(a) < f_\lambda(b)$ pour tout λ compatible avec les informations disponibles sur les préférences du décideur (ses réponses aux questions).

— NSGA-II utilise la crowding distance comme décrit dans la Définition 1.1.21, alors que NEMO définit une crowding distance qui tient compte de l'ensemble des valeurs de paramètres admissibles (les valeurs de λ compatibles avec les données de préférence collectées).

Cet algorithme permet en pratique de se concentrer plus rapidement sur des zones de l'espace de recherche pertinentes pour le décideur que l'algorithme NSGA-II. Par ailleurs, plus on pose de questions, plus la relation de dominance $\prec^N$ permet de discriminer entre les points de la population, et donc plus vite est la convergence vers des solutions pertinentes pour le décideur. Notons que NEMO choisit les questions de sorte à ne pas créer d'incohérence dans la base de données, autrement dit on demande toujours au décideur de comparer deux solutions x, x', telles que $non(y(x) \prec^N y(x'))$ et $non(y(x') \prec^N y(x'))$.

NEMO-II

NEMO-II (pour Necessary-preference-enhanced Evolutionary Multi-objective Optimizer) [Branke et al., 2016] est une amélioration de NEMO dont l'objectif principal est de :

— réduire le temps de calcul nécessaire à l'identification des différents fronts.

— remettre en question le modèle utilisé pour représenter les préférences du décideur.

En effet, dans NEMO, le calcul des fronts nécessite de vérifier si $a \prec^N b$ pour toute paire (a, b) de points dans la population, ce qui nécessite un nombre quadratique de programmes linéaires à résoudre. À la place, dans NEMO-II, on définit les fronts de manière successive en utilisant la notion d'optimalité potentielle : le point a est potentiellement optimal si et seulement s'il existe une valeur λ admissible telle que $f_\lambda(a) < f_\lambda(b)$ pour tout point b de la population. Chaque solution est donc comparée directement à un ensemble de solutions, ce qui peut se faire avec un seul programme linéaire. On passe ainsi d'un nombre quadratique à un nombre linéaire. Les auteurs justifient le passage à cette notion d'optimalité potentielle, non seulement par ce gain en termes de temps de calcul, mais aussi par le fait que cela permette de discriminer plus de solutions qu'avec la relation $\prec^N$. En effet, les non dominés au sens de $\prec^N$ forment un sur-ensemble des points potentiellement optimaux.

En ce qui concerne la remise en question du modèle, cela a été traité de la manière suivante. Cette méthode débute avec un modèle linéaire simple (somme pondérée), puis passe à un modèle plus complexe (intégrale de Choquet) dès que les données de préférences collectées ne peuvent plus être représentées à l'aide de ce premier modèle. Ainsi, contrairement à NEMO qui se restreint aux questions ne contredisant pas le modèle, NEMO-II est plus flexible en autorisant le décideur à se contredire en lui posant des questions concernant des points a et b tel que $a \prec^N b$. Si par la suite le passage à l'intégrale de Choquet ne suffit plus à corriger l'incohérence de la base de données, la procédure élimine les réponses les plus anciennes jusqu'à parvenir à rétablir la cohérence.

NEMO-II se distingue aussi au niveau de la crowding distance, revenant sur la définition utilisée dans NSGA-II, plutôt que celle proposée pour NEMO.

Un pseudo-code de la méthode NEMO-II-Ch, utilisant l'intégrale de Choquet, est disponible ci-dessous, en Algorithme 1. Notons que cette méthode est générale, dans le sens où elle peut s'appliquer à n'importe quel problème d'optimisation multi-objectifs. Dans cet algorithme, une question est posée au décideur toutes les q générations et l'algorithme s'arrête après g générations, où q et g sont des paramètres de l'algorithme.

Dans la section suivante, on s'intéresse à l'élicitation incrémentale fondée sur la notion de regret.

1.3.2 Principe des méthodes fondées sur la notion de regret

Dans cette section, on commence par introduire le critère de décision minimax regret, et on explique comment calculer efficacement les regrets associés, dans le cas où les préférences du décideur sont représentables par une fonction d'agrégation linéaire en ses paramètres. Puis, on présente l'approche d'élicitation incrémentale fondée sur le minimax regret, ainsi qu'une stratégie de génération de questions standard. Enfin, on s'intéresse aux problèmes d'optimisation combinatoire multi-objectifs, en présentant des algorithmes récents combinant résolution et élicitation incrémentale fondée sur le minimax regret.

Le critère de décision minimax regret

Le critère de décision minimax regret a été proposé initialement dans le cadre de problème de décision dans l'incertain [Savage, 1954, Kouvelis and Yu, 2013]. Plus récemment, il a été utilisé dans des situations où l'incertitude (imprécision) porte sur les paramètres de la fonction d'agrégation [Boutilier et al., 2006]. C'est dans ce contexte que nous l'utilisons. On suppose que les préférences du décideur sont représentables par une fonction d'agrégation f_λ dont les paramètres λ sont inconnus par la procédure. On note Θ l'ensemble des informations que nous possédons sur les préférences du décideur. Cet ensemble, potentiellement initialement vide , contient généralement des données de type (a, b) avec $a, b \in \mathbb{R}^p_+$ signifiant que le point a est meilleur que le point b pour le décideur. On note Λ_Θ l'ensemble des paramètres λ compatibles avec les données contenues dans Θ ; l'ensemble Λ_Θ est donc défini par $\Lambda_\Theta = \{\lambda \in [0,1]^p : \forall (a,b) \in \Theta, f_\lambda(a) \leq f_\lambda(b)\}$. Dans le cas où f_λ est une fonction linéaire en ses paramètres, l'ensemble Λ_Θ est un polyèdre convexe.

Algorithme 1 NEMO-II-Ch simple [Branke et al., 2016]

modèle de préférence = linéaire.
Générer une population de solutions initiales.
Poser une question au décideur lui demandant de comparer deux solutions dans la population.
Regrouper les solutions par front en utilisant la notion d'optimalité potentielle.
Ordonner les solutions de chaque front en utilisant la crowding distance.
while le nombre de générations est inférieur à g **do**
    Sélectionner des solutions par tournoi (fondé sur les fronts et la crowding distance).
    Générer des solutions par croisement et mutation, puis les ajouter à la population.
    if on n'a pas posé de questions depuis q itérations **then**
        Demander au décideur de comparer deux solutions de la population.
        if il n'existe plus de valeurs de paramètres compatibles avec les préférences du décideur **then**
            if modèle de préférences = linéaire **then**
                modèle de préférences = Choquet
            else
                Supprimer les réponses aux questions, en commençant par la plus ancienne, jusqu'à ce que la cohérence soit rétablie.
                Réintroduire les réponses dans l'ordre inverse tant que la cohérence est maintenue.
            end if
        end if
    end if
    Regrouper les solutions de la population par front en utilisant l'optimalité potentielle.
    Ordonner les solutions de chaque front en utilisant la crowding distance.
    Réduire la taille de la population à la taille initiale, en éliminant les pires individus (on procède front par front, puis on complète selon la crowding distance).
end while
return la solution du premier front qui possède la plus grande crowding distance.

Le critère de décision minimax regret permet de formuler une recommandation en tenant compte de l'ensemble des paramètres admissibles Λ_Θ. Celui peut être défini à l'aide des notions de regrets suivantes :

Definition 1.3.1 (Pairwise Max Regret)**.** *Le Pairwise Max Regret (PMR) d'une solution $x \in \mathcal{X}$ par rapport à une solution $x' \in \mathcal{X}$ est défini par :*

$$PMR(x, x', \Lambda_\Theta) = \max_{\lambda \in \Lambda_\Theta} \Big\{ f_\lambda(y(x)) - f_\lambda(y(x')) \Big\}$$

La valeur $PMR(x, x', \Lambda_\Theta)$ représente la plus grande perte possible lorsque la solution x est recommandée au décideur à la place de la solution x'.

Definition 1.3.2 (Max Regret)**.** *Le Max Regret (MR) d'une solution $x \in \mathcal{X}$ est défini par :*

$$MR(x, \mathcal{X}, \Lambda_\Theta) = \max_{x' \in \mathcal{X}} PMR(x, x', \Lambda_\Theta)$$

La valeur $MR(x, \mathcal{X}, \Lambda_\Theta)$ indique, dans le pire cas, la perte associée à la recommandation de x plutôt que n'importe quelle autre solution $x' \in \mathcal{X}$. On remarque que, le max regret d'une solution x est toujours positif puisque $PMR(x, x, \Lambda_\Theta) = 0$.

Definition 1.3.3 (Minimax Regret)**.** *Le MiniMax Regret (MMR) est défini par :*

$$MMR(\mathcal{X}, \Lambda_\Theta) = \min_{x \in \mathcal{X}} MR(x, \mathcal{X}, \Lambda_\Theta)$$

La valeur $MMR(\mathcal{X}, \Lambda_\Theta)$ correspond à la plus petite valeur possible de $MR(x, \mathcal{X}, \Lambda_\Theta)$. Les solutions optimales pour le critère de décision minimax regret sont les solutions qui minimisent le max regret, autrement dit $x^* \in \mathcal{X}$ est optimale si et seulement si $MMR(\mathcal{X}, \Lambda_\Theta) = MR(x^*, \mathcal{X}, \Lambda_\Theta)$. Recommander une telle solution permet de garantir que, dans le pire des cas, la perte est minimisée. De plus, si $MMR(\mathcal{X}, \Lambda_\Theta) = 0$, alors on sait que x^* est nécessairement optimale selon les préférences du décideur. En effet, par définition du max regret, on sait que x^* minimise la fonction f_λ pour tous les paramètres admissibles $\lambda \in \Lambda_\Theta$.

Calcul du pairwise max regret par programmation linéaire

Lorsque les préférences du décideur sont représentables par une fonction linéaire en ses paramètres, on peut utiliser la programmation linéaire pour calculer efficacement les pairwise max regrets. C'est le cas de la somme pondérée, d'OWA et de l'intégrale de Choquet, comme décrit dans les articles [Benabbou et al., 2015, Benabbou et al., 2017]. Plus précisément, avec une somme pondérée, le calcul se fait par résolution du programme linéaire suivant :

$$\begin{aligned}
&\max_{\lambda} \sum_{i=1}^{p} (\lambda_i y_i(x) - \lambda_i y_i(x')) && && \\
&\text{s.c.} \sum_{i=1}^{p} \lambda_i = 1 && && (1.4)\\
&\quad \sum_{i=1}^{p} \lambda_i a_i - \sum_{i=1}^{p} \lambda_i b_i \leqslant 0 && \forall (a,b) \in \Theta && (1.5)\\
&\quad \lambda_i \geqslant 0 && \forall i \in \{1, \ldots, p\} && (1.6)
\end{aligned}$$

Dans ce programme linéaire, la contrainte (1.4) impose que les poids soient normalisés, c'est-à-dire qu'ils somment à 1. L'équation (1.5) impose que les informations disponibles sur les préférences du décideur soient respectées : $f_\lambda(a) \leq f_\lambda(b)$ pour toute paire $(a, b) \in \Theta$. L'équation (1.6) impose que les paramètres soient tous positifs. La fonction objectif correspond à la maximisation de l'écart entre $f_\lambda(x)$ et $f_\lambda(x')$.

Avec l'opérateur OWA, le programme linéaire est le suivant :

$$\max_{\lambda} \sum_{i=1}^{p} (\lambda_i y_{(i)}(x) - \lambda_i y_{(i)}(x'))$$

$$\text{s.c.} \sum_{i=1}^{p} \lambda_i = 1 \tag{1.7}$$

$$\lambda_i - \lambda_{i+1} \geqslant 0 \qquad \forall i \in \{1, \ldots, p-1\} \tag{1.8}$$

$$\sum_{i=1}^{p} \lambda_i a_{(i)} - \sum_{i=1}^{p} \lambda_i b_{(i)} \leqslant 0 \qquad \forall (a,b) \in \Theta \tag{1.9}$$

$$\lambda_i \geqslant 0 \qquad \forall i \in \{1, \ldots, p\} \tag{1.10}$$

où (.) représente la permutation des critères permettant de trier les composantes des vecteurs de performance en ordre décroissant. Les contraintes sont similaires à celles de la somme pondérée, mais on ajoute une contrainte supplémentaire sur les paramètres quand on souhaite avoir des poids décroissants (cf. équation (1.8)). En effet, on a vu que les poids doivent être décroissants pour garantir une équité de la solution optimale.

Avec une intégrale de Choquet, puisque $f_\lambda(x)$ est linéaire en λ pour une solution x fixée, le calcul de $PMR(x, x', \Lambda_\Theta)$ peut également être réalisé en résolvant un programme linéaire :

$$\max_{\lambda} \sum_{i=1}^{p} \Big(\big(y_{[i]}(x) - y_{[i-1]}(x)\big)\lambda(v_{X'_{[i]}}) - \big(y_{[i]}(x') - y_{[i-1]}(x')\big)\lambda(v_{X_{[i]}}) \Big)$$

$$s.c.\ v_\emptyset = 0 \tag{1.11}$$

$$v_N = 1 \tag{1.12}$$

$$v_A - v_B \leq 0 \qquad \forall A \subset B \subseteq N \tag{1.13}$$

$$\sum_{i=1}^{p} \Big(\big(a_{[i]} - a_{[i-1]}\big) v_{A_{[i]}} - \big(b_{[i]} - b_{[i-1]}\big) v_{B_{[i]}} \Big) \leq 0 \qquad \forall (a,b) \in \Theta \tag{1.14}$$

$$v_A \geq 0 \qquad \forall A \subseteq N \tag{1.15}$$

où $N = \{1, \ldots, p\}$ est l'ensemble des critères et v_A est la variable représentant la valeur v_A pour tout $A \subseteq N$. De plus (.), $X_{[i]}$, $X'_{[i]}$, $A_{[i]}$, $B_{[i]}$, avec $i \in N$, sont définis suivant la Définition 1.2.5. Les contraintes (1.11) et (1.12) assurent que v est une capacité normalisée, les contraintes de l'équation (1.13) sont les contraintes de monotonie de la capacité et les contraintes (1.14) correspondent aux données de préférence.

Dans ce programme linéaire, le nombre de variables et de contraintes est exponentiel par rapport au nombre de critères. Mais il a été prouvé que ce programme peut être simplifié pour certaines sous-classes d'intégrales de Choquet, en utilisant sa formulation alternative en termes de masses de Möbius. Par exemple, pour la sous-classe des fonctions de croyance,

les contraintes de monotonie $v(A) \leq v(B)$, avec $A \subset B \subseteq N$, sont naturellement satisfaites en raison de la non-négativité des masses de Möbius. Pour la sous-classe des capacités 2-additives, il a été proposé d'utiliser le fait qu'elles forment un polytope convexe avec seulement p^2 points extrêmes. Cela donne une formulation linéaire n'impliquant qu'un nombre quadratique de variables et de contraintes. Dans le cadre de cette thèse, on considérera des fonctions de croyance 2-additives, ce qui conduit à la formulation suivante :

$$\max_{m} \sum_{A\subseteq N:|A|\leq 2} (m_A \min_{i\in A}\{y_i(x)\} - m_A \min_{i\in A}\{y_i(x')\})$$

$$s.c.\ m_\emptyset = 0 \tag{1.16}$$

$$\sum_{i\subseteq N} m_{\{i\}} + \sum_{\{i,j\}\subseteq N} m_{\{i,j\}} = 1 \tag{1.17}$$

$$\sum_{i\subseteq N} m_{\{i\}} \geq 0 \tag{1.18}$$

$$m_{\{i\}} + \sum_{j\subseteq T} m_{\{i,j\}} \geq 0 \qquad \forall i \in N, \forall\, T \subseteq A\backslash\{i\}, T \neq \emptyset \tag{1.19}$$

$$\sum_{A\subseteq N:|A|\leq 2} m_A \min_{i\in A}\{a_i\} - \sum_{A\subseteq N:|A|\leq 2} m_A \min_{i\in A}\{b_i\} \leq 0 \qquad \forall (a,b) \in \Theta \tag{1.20}$$

où m_A est la variable représentant la valeur $m(A)$ pour tout $A \subseteq N$ avec $|A| \leq 2$. Les contraintes (1.16) et (1.17) sont des contraintes de normalisation, les contraintes (1.18) et (1.19) sont celles de monotonie et les contraintes (1.20) correspondent à la compatibilité avec les préférences observées.

Néanmoins, des résultats positifs ont été obtenus pour le cas des capacités générales, lorsque l'ensemble Θ contenant les informations sur les préférences possède une forme particulière. Plus précisément, dans l'article [Benabbou et al., 2014], il a été montré que si Θ contient uniquement des paires composées :

- d'une alternative binaire de type $1A0$, représentant une solution fictive avec une performance égale à 1 sur les critères dans A et égale à 0 dans $N\backslash A$.
- d'une alternative à profil constant de type $(\mu, \ldots, \mu)$, avec $\mu \in [0, 1]$, représentant une solution fictive avec une performance identique sur tous les critères.

alors, les PMRs peuvent être formulés en un programme linéaire avec un nombre de contraintes et de variables linéaires. Dans le même article, les auteurs proposent une stratégie de génération de questions permettant de collecter des données de préférences de la bonne forme, et qui est très efficace en pratique pour réduire les valeurs de minimax regret. La stratégie de génération de questions proposée peut être adaptée pour assurer un nombre polynomial de questions, en sélectionnant l'alternative à profil constant au milieu de l'intervalle représentant les valeurs de capacités possibles [Benabbou and Perny, 2017].

Calcul du minimax regret avec élagage alpha-bêta

Dans le paragraphe précédent, nous avons vu comment calculer les PMRs selon la fonction d'agrégation choisie. En suivant les Définitions 1.3.2 et 1.3.3, on peut alors obtenir la valeur MMR en calculant les PMRs associés à toutes les paires de solutions $(x, x') \in \mathcal{X}$. Cependant, cette approche nécessite de résoudre un nombre quadratique de problèmes d'optimisation. En pratique, il est possible de réduire considérablement le nombre de calculs de PMR nécessaires, avec des élagages de type alpha-bêta, qui sont utilisés classiquement dans les problèmes de type min-max. En effet, lors du calcul du MMR, on peut calculer les MRs associés aux différentes solutions de manière itérative, en conservant le plus petit MR obtenu à tout instant (noté MMR-courant). Pour calculer le MR d'une alternative x donnée, on peut calculer le PMR de x par rapport à toutes les autres solutions $x' \in \mathcal{X}$ de manière itérative, puis récupérer à la fin le plus grand PMR obtenu. On peut procéder à un élagage lorsque la valeur $PMR(x, x', \Lambda_\Theta)$ pour une alternative $x' \in \mathcal{X}$ donnée est supérieure à la valeur MMR-courant, puisque cela implique que $MR(x, \Lambda_\Theta)$ est plus grand que le plus petit MR trouvé jusque-là. Dans ce cas, on sait que x n'est pas une solution optimale pour le critère minimax regret, et on peut passer à la solution suivante. Cet élagage permet en pratique de réduire considérablement les temps d'exécution. L'exemple suivant permet d'illustrer ce calcul :

Exemple 1.3.2. *On considère un problème impliquant quatre solutions* $\mathcal{X} = \{s_1, s_2, s_3\}$ *évaluées sur trois critères de la manière suivante :* $\mathcal{Y} = \{y(s_1) = (5, 2, 4), y(s_2) = (2, 9, 6), y(s_3) = (7, 6, 3)\}$. *On suppose ici que les préférences du décideur sont représentables par une somme pondérée* f_λ *dont le jeu de poids* $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ *est inconnu du système. Néanmoins, on possède les données de préférences suivantes :* $\Theta = \{((5, 2, 4), (2, 9, 6))\}$ *(obtenues par exemple en demandant directement au décideur s'il préfère la solution* s_1 *à la solution* s_2*). De ce fait, l'ensemble des paramètres admissibles* $\Lambda_\Theta = \{\lambda : f_\lambda(s_1) \leq f_\lambda(s_2)\} = \{\lambda : \lambda_2 \geq \frac{3}{7}\lambda_1 - \frac{2}{7}\lambda_3\}$.

On souhaite ici calculer le minimax regret avec élagage alpha-bêta et identifier la solution optimale au sens de ce critère. Pour calculer le MMR, on calcule itérativement les MR de s_1, s_2, s_3, *dans cet ordre. Pour le calcul de chaque MR, on utilise la programmation linéaire pour calculer les différents PMRs, et on interrompt éventuellement le calcul du MR en suivant l'élagage alpha-bêta décrit ci-dessus. On obtient la suite de calculs suivante :*

Calcul de $MR(s_1, \mathcal{X}, \Lambda_\Theta)$ *:*

— $PMR(s_1, s_1) = 0$ *(pas besoin de calculer en pratique).*

— $PMR(s_1, s_2) = 0$.

— $PMR(s_1, s_3) = 1$.

Le max regret de la solution s_1 *vaut donc* $MR(s_1, \mathcal{X}, \Lambda_\Theta) = 1$, *et on conserve l'information que le plus petit MR trouvé jusqu'à présent est égal à 1 en posant MMR-courant* $= 1$. *On lance maintenant les calculs pour le MR de* s_2 *:*

— $PMR(s_2, s_1) = 7$.

Comme $PMR(s_2, s_1) = 7 \geq$ *MMR-courant, alors on arrête le calcul du MR de* s_2 *puisque cela implique que* s_2 *n'est pas optimale au sens du minimax regret. On passe aux calculs associés au MR de* s_3 *:*

— $PMR(s_3, s_1) = 4$.

Comme $PMR(s_3, s_1) = 4 \geq$ *MMR-courant, alors on arrête ici aussi le calcul.*

Ainsi, on a $MMR(\mathcal{X}, \Lambda_\Theta) = 1$ *et* s_1 *est la solution optimale pour le critère minimax regret. Remarquons que, grâce à l'élagage, le minimax regret a été ici déterminé en résolvant* 4 *programmes linéaires (contre* 6 *sans élagage).*

Élicitation incrémentale et stratégies standards

Lorsque la valeur $MMR(\mathcal{X}, \Lambda_\Theta)$ est trop grande, recommander une solution optimale au sens du minimax regret peut induire une perte trop élevé pour le décideur. Dans ces situations, il est préférable de collecter des informations sur les préférences du décideur, afin de réduire la valeur du minimax regret, et donc la perte dans le pire cas. En effet, on sait que $MMR(\mathcal{X}, \Lambda_{\Theta'}) \leq MMR(\mathcal{X}, \Lambda_\Theta)$ pour tout ensemble $\Theta' \supseteq \Theta$, comme souligné dans d'autres travaux (voir [Benabbou et al., 2017] par exemple). En d'autres termes, cela signifie que la valeur du minimax regret diminue en posant des questions au décideur. Ainsi, le principe de l'élicitation incrémentale fondée sur les regrets est de collecter des informations sur les préférences du décideur, en lui posant des questions de manière itérative, jusqu'à ce que la valeur $MMR(\mathcal{X}, \Lambda_\Theta)$ devienne inférieure à un seuil de tolérance donné $\delta \geq 0$ représentant la perte maximale admissible. Si nous prenons $\delta = 0$, alors on est sûr d'obtenir la solution optimale pour le décideur à la fin de l'exécution.

Pour mettre en œuvre cette approche, il faut définir une stratégie de génération de questions permettant de diminuer rapidement la valeur $MMR(\mathcal{X}, \Lambda_\Theta)$. Différentes stratégies ont été proposées dans la littérature, comme le "worst-case maximum minimax regret" qui consiste à choisir deux solutions à comparer, parmi celles de l'ensemble $\mathcal{X}$, qui permettent de réduire le plus possible la valeur du minimax regret dans le pire scénario de réponse [Benabbou, 2017]. Cette stratégie étant trop coûteuse en pratique, il a été proposé d'utiliser une heuristique permettant de réduire efficacement le minimax regret en pratique. Cette heuristique, appelée Current Solution Strategy (CSS) [Boutilier et al., 2006, Drummond and Boutilier, 2013], consiste à demander au décideur de comparer à chaque itération les deux solutions suivantes : une solution x^* optimale au sens du minimax regret et une solution $\hat{x}$ induisant la plus grande perte quand on recommande x^* dans le pire des scénarios (on dit que $\hat{x}$ est un des pires adversaires de x^*). Formellement, x^* et $\hat{x}$ sont définies de la manière suivante :

$$x^* \in \arg\min_{x \in \mathcal{X}} MR(x, \mathcal{X}, \Lambda_\Theta) \tag{1.21}$$

$$\hat{x} \in \arg\max_{x \in \mathcal{X}} PMR(x^*, x, \Lambda_\Theta) \tag{1.22}$$

Avec cette stratégie, les questions sont très souvent informatives, en ce sens qu'elles nous permettent de réduire considérablement l'espace des paramètres admissibles, et donc l'ensemble des solutions potentiellement optimales pour le décideur. Plus une question est informative, moins il reste d'alternatives à départager aux itérations suivantes.

Un autre avantage de cette approche, prouvé dans l'article de Benabbou et al [Benabbou et al., 2015], est que la stratégie CSS ne produit pas d'incohérence, autrement

dit l'ensemble des paramètres admissibles Λ_{Theta} ne peut devenir vide suite à l'ajout des réponses du décideur dans l'ensemble Θ.

Notons toutefois qu'empêcher les incohérences présente tout de même un inconvénient. En effet, dans ce cas, une erreur du décideur devient indétectable. Une fois la mauvaise réponse donnée, il est alors impossible de faire marche arrière, la recherche se concentrant alors sur une mauvaise zone de l'espace des paramètres.

On présente ci-dessous un exemple d'utilisation de la CSS.

Exemple 1.3.3. *On considère un problème avec trois critères et quatre alternatives. Plus précisément, on a ici* $\mathcal{X} = \{s_1, s_2, s_3, s_4\}$ *et* $\mathcal{Y} = \{y(s_1) = (5, 2, 4), y(s_2) = (2, 9, 6), y(s_3) = (7, 6, 3), y(s_4) = (3, 6, 8)\}$. *Les préférences du décideur sont représentables par une somme pondérée* f_λ *avec* $\lambda^* = (0.3, 0.5, 0.2)$. *Les paramètres sont inconnus de la procédure CSS et servent ici uniquement à simuler les réponses du décideur. Le polyèdre des jeux de poids admissibles* Λ_Θ *est donc composé des jeux de poids* $\lambda = (\lambda_1, \lambda_2, \lambda_3) \in [0, 1]^3$ *tels que* $\lambda_1 + \lambda_2 + \lambda_3 = 1$ *(on considère ici que* Θ *est initialement vide). Les points extrêmes de ce polyèdre sont* $(1, 0, 0)$, $(0, 1, 0)$ *et* $(0, 0, 1)$ *(représentés en Figure 1.20). L'objectif ici est d'identifier la solution optimale pour le décideur en utilisant la CSS, et donc il s'agit d'appliquer la méthode avec* $\delta = 0$.

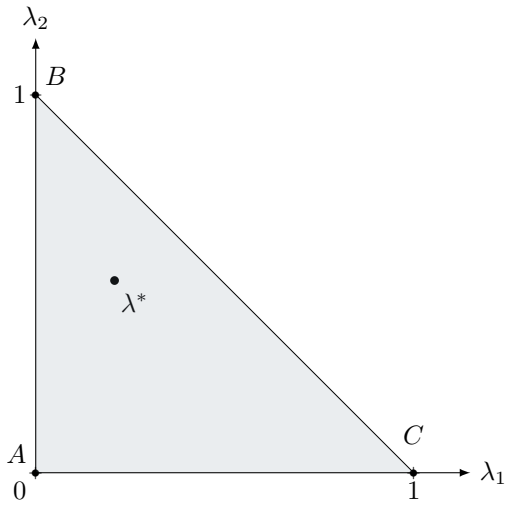


FIGURE 1.20 – Paramètres admissibles Λ_Θ au début de la CSS.

Première itération : comme on a $MMR(\mathcal{X}, \Lambda_\Theta) = 3 > \delta = 0$, *alors il faut demander au décideur de comparer une solution optimale pour le minimax regret* x^* *et un de ses pires adversaires* $\hat{x}$. *Comme* $MMR(\mathcal{X}, \Lambda_\Theta) = MR(s_1, \mathcal{X}, \Lambda_\Theta) = PMR(s_1, s_2, \Lambda_\Theta)$, *on a* $x^* = s_1$ *et* $\hat{x} = s_2$ *(cf. équations 1.21 et 1.22). Le décideur nous répond qu'il préfère* s_1 *à* s_2 *puisque* $f_{\lambda^*}(s_1) = 3.7 < 6.9 = f_{\lambda^*}(s_2)$. *L'ensemble* Θ *des préférences collectées est alors mis à jour :* $\Theta = \{((5, 2, 4), (2, 9, 6))\}$. *L'ensemble des paramètres admissibles est ainsi implicitement actualisé : on impose* $f_\lambda(s_1) \leq f_\lambda(s_2)$, *c'est-à-dire* $5\lambda_1 + 2\lambda_2 + 4\lambda_3 \leq 2\lambda_1 + 9\lambda_2 + 6\lambda_3$, *et donc* $\lambda_2 \geq \lambda_1 - \frac{2}{5}$. *On se retrouve alors avec le polyèdre défini par les points* $A = (0, 0)$, $B = (0, 1)$,

$D = (0.7, 0.3)$ *et* $E = (0.4, 0)$*, représenté en Figure 1.21. Le triangle CDE correspond quant à lui aux jeux de poids qui ne sont plus admissibles.*

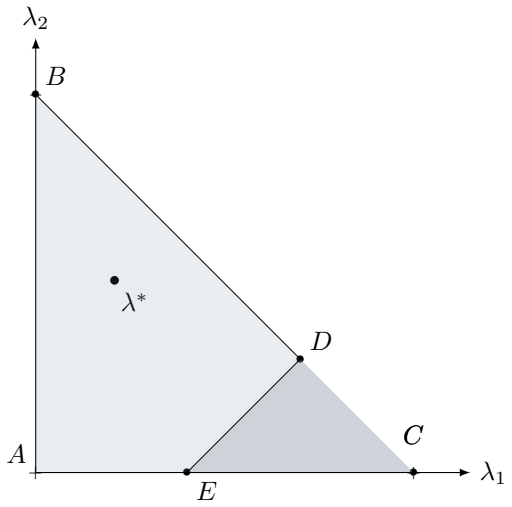


FIGURE 1.21 – Paramètres admissibles Λ_Θ après une question.

Deuxième itération : puisque $MMR(\mathcal{X}, \Lambda_\Theta) = 1 > \delta = 0$*, selon la CSS, on doit donc poser une nouvelle question au décideur. En suivant les équations 1.21 et 1.22, on obtient* $x^* = s_1$ *et* $\hat{x} = s_3$*. Le décideur nous répond qu'il préfère* s_1 *puisque* $f_{\lambda^*}(s_1) = 3.7 < 6 = f_{\lambda^*}(s_3)$*. L'ensemble des préférences collectées est alors mis à jour, soit* $\Theta = \{((5, 2, 4), (2, 9, 6)), ((5, 2, 4), (7, 6, 3))\}$*. L'ensemble des paramètres admissibles doit également être actualisé avec la contrainte* $f_\lambda(s_1) \leq f_\lambda(s_3)$*, c'est-à-dire* $\lambda_2 \geq \frac{1}{5} - \frac{3}{5}\lambda_1$*. Les points extrêmes du polyèdre des jeux de poids admissibles* λ_Θ *deviennent alors* $B = (0, 1)$*,* $D = (0.7, 0.3)$*,* $E = (0.4, 0)$*,* $F = (0, 0.2)$ *et* $G = (\frac{1}{3}, 0)$*, représentés sur la Figure 1.22.*

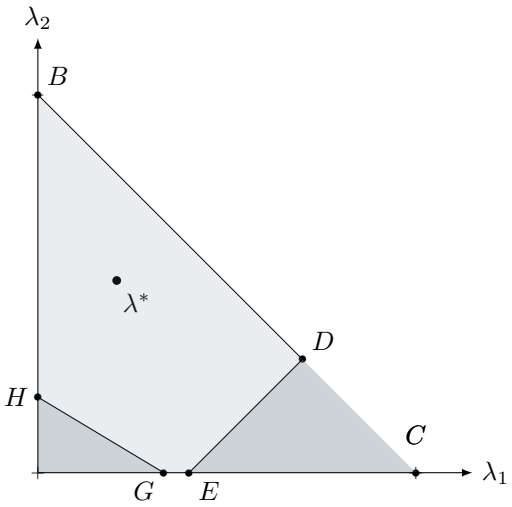


FIGURE 1.22 – Paramètres admissibles Λ_Θ après deux questions.

Troisième itération : on a $MMR(\mathcal{X}, \Lambda_\Theta) = 0 \leq \delta$. Par conséquent, la procédure s'interrompt et retourne la solution s_1 au décideur, avec la certitude que c'est la solution optimale de ce problème. On remarque que la procédure n'a eu besoin que de deux questions pour identifier la meilleure solution parmi un ensemble composé de quatre solutions. De plus, on est sûr de retourner la solution optimale, malgré que l'espace des paramètres admissibles est encore très grand à la fin de la procédure. Cependant, cet exemple illustre aussi le côté conservateur du critère minimax regret, car la solution s_1 avait été trouvée à la première itération, mais considérer le pire scénario nous a conduits à poser deux questions avant de se décider à la retourner. Néanmoins, le seuil de tolérance δ permet de contourner indirectement cette difficulté, en autorisant de retourner une solution avec une perte bornée par δ. Par exemple, en prenant ici $\delta = 2$, on se serait arrêté après seulement une question.

Cette approche a principalement été étudiée dans le cadre de problèmes non combinatoires, car le calcul de $MMR(\mathcal{X}, \Lambda_\Theta)$ nécessite de considérer toutes les solutions possibles, de manière explicite ou implicite, ce qui peut être une tâche coûteuse. Néanmoins, ce problème a fait l'objet de contributions récentes, présentées dans la section suivante.

1.3.3 Méthodes fondées sur la notion de regret sur domaine combinatoire

Pour appliquer la stratégie CSS aux problèmes d'optimisation combinatoire multi-objectifs, on pourrait utiliser la méthode en deux phases suivante : 1) calculer l'ensemble des solutions efficaces, 2) appliquer la CSS sur cet ensemble. Cependant, cette méthode est très coûteuse en pratique, car le nombre de solutions efficaces peut être exponentiel en la taille du problème. À la place, il a été proposé récemment d'utiliser une nouvelle approche, combinant recherche et élicitation incrémentale fondée sur la notion de regret.

Approche générale combinant résolution et élicitation

Cette nouvelle approche consiste à intégrer l'élicitation incrémentale fondée sur la notion de regret à la résolution du problème d'optimisation combinatoire [Benabbou, 2017]. Elle est mise en œuvre de la manière suivante : 1) conception d'un algorithme efficace pour la recherche des solutions potentiellement optimales, 2) identification des moments clés où il faut poser des questions au décideur pour garantir de retourner une solution nécessairement optimale à la fin de l'exécution. En pratique, cette méthode réalise des calculs de regrets sur des sous-ensembles de solutions, ce qui permet de contourner la difficulté rencontrée par la méthode en deux phases. Comme souligné dans l'article de Benabbou et Perny [Benabbou and Perny, 2018], l'idée principale derrière cette nouvelle approche est de :

— réduire le temps de résolution, en utilisant les réponses du décideur pour diriger la recherche vers les solutions les plus prometteuses rapidement,

— poser uniquement les questions nécessaires à discriminer les solutions rencontrées pendant la recherche.

Cette nouvelle approche a été principalement appliquée dans le cadre de la résolution de problèmes où les préférences du décideur sont représentables par une somme pondérée (avec une exception notable dans les graphes d'états avec l'intégrale de Choquet [Benabbou and Perny, 2015a]). Dans ces travaux, les méthodes proposées sont exactes et permettent donc de recommander une solution avec des garanties de performances sur la qualité de la solution (définies à partir du seuil de tolérance δ). Parmi les outils employés, on peut citer la programmation dynamique, le branch and bound, et d'autres approches exactes (par exemple, la recherche gloutonne). Cette approche s'est montrée particulièrement efficace pour réduire les temps de calculs et le nombre de questions nécessaire à formuler une bonne recommandation. Elle a été utilisée dans le cadre de problème de décision multicritère [Benabbou and Perny, 2015b, Benabbou and Perny, 2015a], multi-agents [Benabbou and Perny, 2016, Benabbou et al., 2016], et dans l'incertain [Benabbou and Perny, 2017]. Dans le paragraphe suivant, nous présentons un algorithme de résolution interactif particulier qui suit cette approche générale.

Élicitation incrémentale basée sur les points extrêmes (IEEP)

La méthode IEEP (pour "Incremental Elicitation based on Extreme Points" en anglais)est une méthode de résolution générale combinant recherche et élicitation incrémentale fondée sur les regrets. Elle peut être appliquée à tout problème d'optimisation combinatoire multi-objectifs, pourvu que la fonction d'agrégation soit linéaire en ses paramètres et qu'il existe un algorithme de résolution efficace quand les paramètres sont connus [Benabbou and Lust, 2019a, Benabbou and Lust, 2019b]. En effet, une méthode de résolution est utilisée pour obtenir la solution optimale correspondant à chaque point extrême du polyèdre représentant les valeurs des paramètres admissibles. Cette méthode interactive possède la garantie de retourner une solution avec un regret maximum inférieur à un seuil donné $\delta \geq 0$, à condition que des méthodes de résolution exactes soient utilisées pour chaque point extrême du polyèdre (et non des heuristiques). Cette méthode est décrite dans l'algorithme 2. Dans cet algorithme, le choix des solutions x et x' constituant la question peut être dicté par différentes stratégies de questions ; les auteurs ont notamment comparé la stratégie CSS avec une stratégie de sélection consistant à choisir les solutions x et x' qui sont les plus éloignées au sens de la distance euclidienne.

Une limite de cette méthode est son utilisation des points extrêmes du polyèdre, parce qu'ils peuvent être très nombreux. En particulier, pour le problème du voyageur de commerce, nous verrons que son temps d'exécution augmente de manière exponentielle avec une somme pondérée, et que pour l'intégrale de Choquet, l'algorithme ne se termine pas en temps raisonnable dès que $p \geq 4$ (cf. chapitre 2).

Adaptation avec une approche bayésienne

Récemment, il a été proposé d'adapter cette approche pour gérer des possibles incohérences provenant des réponses du décideur, en remplaçant le critère minimax regret par un regret espéré [Bourdache et al., 2020]. L'incertitude sur les paramètres du modèle est alors

Algorithme 2 `IEEP`

Générer les points extrêmes du polyèdre Λ_Θ
Générer l'ensemble X_Θ composé d'une solution optimale par point extrême du polyèdre
while $MMR(X_\Theta, \Lambda_\Theta) > \delta$ **do**
  Sélectionner de deux solutions x, x' dans l'ensemble X_Θ
  Demander au décideur de comparer x et x'
  Ajouter la réponse dans l'ensemble Θ
  Mettre à jour (implicitement) le polyèdre : $\Lambda_\Theta = \{\lambda : \forall (a,b) \in \Theta, f_\lambda(a) \leq f_\lambda(b)\}$
  Générer les points extrêmes du polyèdre Λ_Θ
  Générer l'ensemble X_Θ composé d'une solution optimale par point extrême du polyèdre
end while
return une solution optimale au sens du minimax regret dans l'ensemble X_Θ

représentée par une fonction de densité sur l'espace des paramètres et les réponses aux questions sont utilisées pour réduire cette incertitude par révision bayésienne.

La stratégie de sélection des questions est basée sur la résolution d'un programme linéaire en nombres entiers, avec un ensemble combinatoire de variables et de contraintes. De ce fait, il a été proposé d'utiliser des méthodes de génération de colonnes et de contraintes pour pouvoir utiliser cette méthode en pratique.

Bien que remplacer le minimax regret par un regret espéré permette de gérer les possibles erreurs du décideur, les calculs des regrets espérés ainsi que la révision bayésienne sont des opérations très coûteuses en pratique. De plus, permettre au décideur de se contredire ne permet plus de garantir la convergence de la procédure.

Positionnement de la thèse par rapport à ces approches

Dans le cadre de cette thèse, nous adoptons la même approche consistant à combiner résolution et élicitation fondée sur les regrets. Cette approche est en effet très intéressante, car elle permet à la fois de réduire les temps de calcul, en évitant d'explorer tout l'espace des solutions, et de réduire le nombre de questions posées, en se concentrant sur les solutions rencontrées durant l'exploration. L'originalité de cette thèse réside dans l'utilisation de métaheuristiques comme algorithme de résolution, ce qui permet de traiter des problèmes plus complexes. La complexité peut provenir de la considération de modèles de préférences plus sophistiqués, comme l'intégrale de Choquet. L'utilisation de métaheuristiques permet aussi de traiter des problèmes d'optimisation combinatoire multi-objectifs de nature plus complexe, autrement dit ceux pour lesquels on ne connaît pas d'algorithme de résolution exacte (ou approché avec garantie) dans le cas où les préférences sont parfaitement connues. On verra dans les chapitres 3 et 4 qu'un autre apport de cette thèse concerne l'élicitation incrémentale combinée avec l'option dans les matroïdes.

Chapitre 2

Métaheuristiques interactives pour la résolution de problèmes d'optimisation combinatoire multi-objectifs

Résumé

Dans ce chapitre, nous proposons deux métaheuristiques interactives permettant la résolution de problèmes d'optimisation combinatoire multi-objectifs avec préférences imprécises. Plus précisément, nous considérons des problèmes où les préférences du décideur sur les solutions peuvent être représentées par une fonction d'agrégation paramétrée, et nous supposons que les paramètres ne sont initialement pas connus par le système de recommandation. Pour déterminer la meilleure solution pour le décideur, on utilise une approche fondée sur la recherche heuristique couplée à l'élicitation incrémentale. L'imprécision des paramètres représentant les préférences du décideur est progressivement réduite en posant des questions sur ses préférences de manière itérative durant la recherche. Ainsi, nous avons conçu des algorithmes de recherche locale et génétique interactifs permettant de résoudre tout problème d'optimisation combinatoire multi-objectifs, sous les conditions suivantes : la fonction de scalarisation doit être linéaire en ses paramètres (comme une somme pondérée, un agrégateur OWA, une intégrale de Choquet), et une solution (quasi-)optimale doit pouvoir être déterminée efficacement lorsque les paramètres sont connus avec précision. Afin de démontrer l'efficacité pratique de notre approche, nos algorithmes sont testés numériquement sur le problème du voyageur de commerce et du sac à dos, pour des préférences représentées par une somme pondérée, puis par un agrégateur de type OWA et enfin une intégrale de Choquet 2-additive, dans le but de démontrer son adaptabilité. Les travaux présentés dans ce chapitre ont fait l'objet de trois publications [Benabbou et al., 2019, Benabbou et al., 2020a, Benabbou et al., 2020b].

2.1 Deux approches générales fondées sur le minimax regret

La sous-section suivante présente notre premier algorithme interactif basé sur la recherche locale. Développée dans le chapitre précédent (cf. Section 1.1.4), la recherche locale est une heuristique qui consiste à passer d'une solution réalisable à une autre par améliorations successives. L'élicitation va nous permettre d'identifier ces améliorations.

2.1.1 Combiner recherche locale et élicitation incrémentale

Une recherche locale explore un voisinage de la solution courante puis sélectionne la meilleure solution dans ce voisinage pour l'itération suivante. Le processus peut par exemple s'arrêter lorsqu'il n'est plus possible d'améliorer la solution courante, lorsque le nombre d'itérations choisi est atteint ou encore lorsque le temps imparti est dépassé.

Dans les problèmes que nous considérons, les préférences du décideur ne sont pas connues de manière précise. De ce fait, il n'est pas toujours possible d'identifier la solution qu'il préfère dans un voisinage. Ainsi, nous proposons de combiner la recherche locale à l'élicitation incrémentale, en posant des questions au décideur durant la recherche afin de pouvoir sélectionner des solutions pertinentes dans les voisinages. On posera aussi des questions durant la phase initiale, afin d'identifier une bonne solution de départ pour notre algorithme de recherche locale incrémentale.

Notre algorithme prend en entrée un problème d'optimisation combinatoire multi-objectifs, deux seuils de tolérance $\delta_1, \delta_2 \geq 0$, une fonction scalarisante f_λ dont les paramètres λ sont inconnus et un ensemble de données de préférences Θ (potentiellement vide). Avant d'entamer la recherche locale interactive, l'algorithme recherche un bon point de départ en procédant comme suit :

1. On génère aléatoirement m jeux de paramètres, notés λ^k avec $k \in \{1, \ldots, m\}$.
2. Pour tout $k \in \{1, \ldots, m\}$, on détermine une solution (quasi-)optimale pour la fonction f_{λ^k} en utilisant un algorithme efficace pour le problème avec modèle de préférence connu. On note $\mathcal{X}_0$ l'ensemble de ces m solutions.
3. Finalement, on collecte des informations sur les préférences du décideur pour déterminer le meilleur point de départ parmi les solutions de l'ensemble $\mathcal{X}_0$. Pour ce faire, on procède comme suit : tant que $MMR(\mathcal{X}_0, \Lambda_\Theta) > \delta_1$, on demande au décideur de comparer deux solutions x, x' de l'ensemble $\mathcal{X}_0$. Cette question permet de mettre à jour l'ensemble des paramètres admissibles en insérant la contrainte linéaire $f_\lambda(x) \leq f_\lambda(x')$ (ou l'inverse selon sa réponse). Une fois que l'on obtient $MMR(\mathcal{X}_0, \Lambda_\Theta) \leq \delta_1$, on choisit comme point de départ une solution optimale pour le minimax regret, c'est-à-dire une solution x^* arbitrairement choisie dans l'ensemble $\arg\min_{x \in \mathcal{X}} MR(x, \mathcal{X}_0, \Lambda_\Theta)$.

Ensuite, notre algorithme réalise une recherche locale interactive en passant de solutions en solutions par exploration de voisinages, à partir de la solution de départ x^*. Plus précisément, notre algorithme réalise les étapes suivantes :

1. À partir de la solution x^*, on engendre un ensemble de solutions "voisines", noté $\mathcal{X}^*$, à l'aide d'une fonction de voisinage spécifique au problème. La solution x^* est également ajoutée à l'ensemble $\mathcal{X}^*$. Puis on enlève de cet ensemble toute solution Pareto-dominée par une autre.
2. On cherche ensuite à déterminer la meilleure solution de l'ensemble $\mathcal{X}^*$ afin de définir la nouvelle solution courante (c'est-à-dire la solution à laquelle sera appliquée la fonction de voisinage à la prochaine itération). Ainsi, tant que $MMR(\mathcal{X}^*, \Lambda_\Theta) > \delta_2$, on demande au décideur de comparer deux solutions x, x' de l'ensemble $\mathcal{X}^*$ et l'ensemble Λ_Θ est réduit par l'insertion de la contrainte $f_\lambda(x) \leq f_\lambda(x')$ (ou inversement).
3. Enfin, la solution x^* est remplacée par une solution optimale pour le minimax regret si et seulement si $MR(x^*, \mathcal{X}^*, \Lambda_\Theta) > \delta_2$. Dans ce cas, une solution est arbitrairement choisie dans $\arg\min_{x \in \mathcal{X}^*} MR(x, \mathcal{X}^*, \Lambda_\Theta)$. L'algorithme s'arrête si la solution courante est la même qu'à l'étape précédente (on a atteint un minimum local). La solution finale est retournée après au plus it_{max} itérations.

Notre algorithme, nommé ILS pour *Interactive Locale Search*, est décrit en Algorithme 3. Un exemple d'exécution est donné dans la Section 2.2.2 sur un problème particulier (le problème du voyageur de commerce). Les lignes 1 à 9 ont pour objectif de déterminer la première solution courante de la recherche locale et les lignes 11 à 29 correspondent à la procédure de recherche locale interactive.

Par ailleurs, l'exemple ci dessous permet de montrer l'importance de l'étape de sélection du point de départ de la recherche locale.

Exemple 2.1.1. *Plus précisément, la Figure 2.1 représente l'exécution de notre algorithme sur un problème du voyageur de commerce bicritère de 100 villes. Ainsi, les points en noir représentent le front de Pareto du problème.*

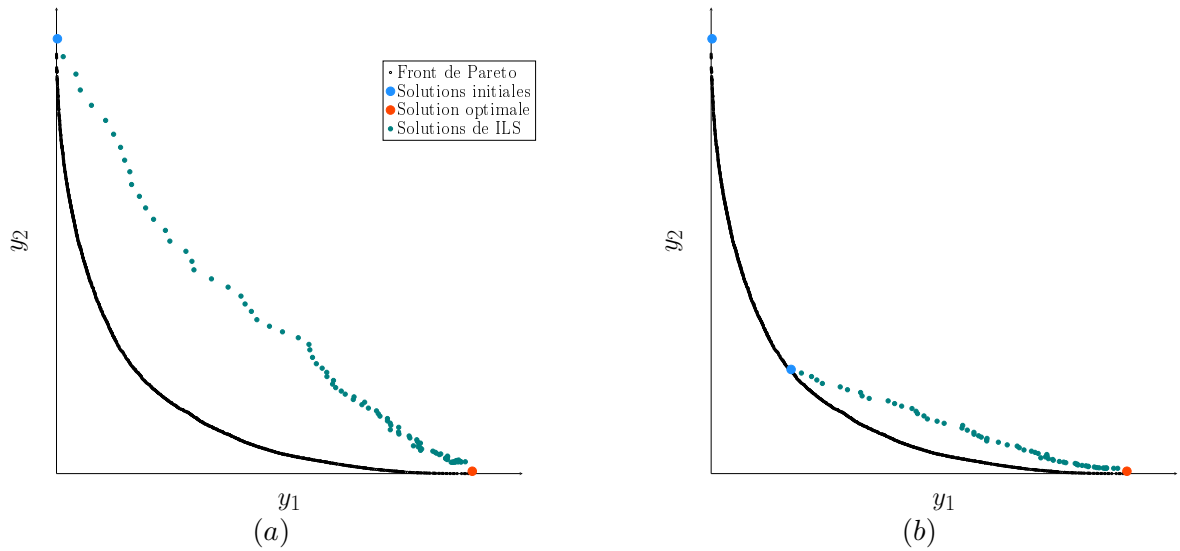


FIGURE 2.1 – Recherche locale sur une instance avec points de départ différents.

En Figure 2.1(a) on a $m = 1$ et en Figure 2.1(b) $m = 2$ solutions potentielles pour être le point de départ (représentées par des points bleus sur la figure). Chaque point en vert correspond à une solution courante explorée durant l'exécution de la recherche locale interactive (cela correspond au nombre d'itérations fait par la recherche locale)

Sur la Figure 2.1(a), on observe que le nombre d'itérations peut être très grand si le point de départ est très loin de la solution optimale. Sur la Figure 2.1(b), on voit qu'augmenter le nombre de points de départ possibles permet de se rapprocher de la solution optimale et donc de réduire le nombre d'itérations.

Algorithme 3 `Interactive local search (ILS)`

Initialisation des paramètres admissibles : $\Lambda_\Theta \leftarrow \{\lambda : \forall (a,b) \in \Theta, f_\lambda(a) \leq f_\lambda(b)\}$
Générations des m solutions initiales : $\mathcal{X}_0 \leftarrow$ `Optimisation`(Λ_Θ, m)
Détermination de la solution de départ :
while $MMR(\mathcal{X}_0, \Lambda_\Theta) > \delta_1$ **do**
  $(x, x') \leftarrow$ `Question`$(\mathcal{X}_0)$
  Mise à jour des préférences connues :
  $\Theta \leftarrow \Theta \cup \{x, x'\}$
  $\Lambda_\Theta \leftarrow \{\lambda : \forall (a,b) \in \Theta, f_\lambda(a) \leq f_\lambda(b)\}$
end while
$x^* \leftarrow \arg\min_{x \in \mathcal{X}_0} MR(x, \mathcal{X}_0, \Lambda_\Theta)$
Lancement de la recherche locale interactive à partir de x^* :
amélioration $\leftarrow$ **true**
$it \leftarrow 0$
while amélioration and $it \leq it_{max}$ **do**
  $\mathcal{X}^* \leftarrow$ `Voisinage`$(x^*) \cup \{x^*\}$
  while $MMR(\mathcal{X}^*, \Lambda_\Theta) > \delta_2$ **do**
    $(x, x') \leftarrow$ `Question`$(\mathcal{X}^*)$
    Mise à jour des préférences connues :
    $\Theta \leftarrow \Theta \cup \{(x, x')\}$
    $\Lambda_\Theta \leftarrow \{\lambda : \forall (a,b) \in \Theta, f_\lambda(a) \leq f_\lambda(b)\}$
  end while
  Choix de la solution suivante :
  if $MR(x^*, \mathcal{X}^*, \Lambda_\Theta) > \delta_2$ **then**
    $x^* \leftarrow \arg\min_{x \in \mathcal{X}^*} MR(x, \mathcal{X}^*, \Lambda_\Theta)$
    $it \leftarrow it + 1$
  else
    amélioration $\leftarrow$ **false**
  end if
end while
return x^*

On note que les procédures `Optimisation` et `Voisinage` dépendent du problème considéré. Pour la détermination d'un voisinage, plusieurs fonctions peuvent être employées comme

nous le verrons dans la suite de cette section sur le problème du voyageur de commerce par exemple.

La recherche locale se trouvant être efficace en pratique, la seconde heuristique que nous avons étudiée est l'approche génétique dans le but d'améliorer nos résultats. La section suivante décrit cet algorithme.

2.1.2 Combiner algorithme génétique et élicitation incrémentale

Comme décrit dans le chapitre précédent (cf. Section 1.1.3), les algorithmes génétiques s'inspirent de phénomènes biologiques pour faire évoluer une population de solutions réalisables de sorte à faire émerger une solution (quasi-)optimale. On rappelle brièvement que le principe général d'un tel algorithme est de commencer par évaluer les solutions d'une population initiale et d'en sélectionner les meilleures. Puis, à partir de ces dernières solutions, on imite un brassage génétique en créant de nouvelles solutions par croisements et mutations de solutions. On obtient ainsi une nouvelle population sur laquelle on itère les étapes précédentes, de manière à simuler le principe de l'évolution. On peut interrompre le processus après un nombre arbitraire d'itérations (aussi appelées générations) ou après l'identification d'une solution de qualité satisfaisante ou encore après un temps imparti. La Figure 2.2 montre l'évolution d'une population qui se déplace progressivement vers la solution optimale pour le décideur.

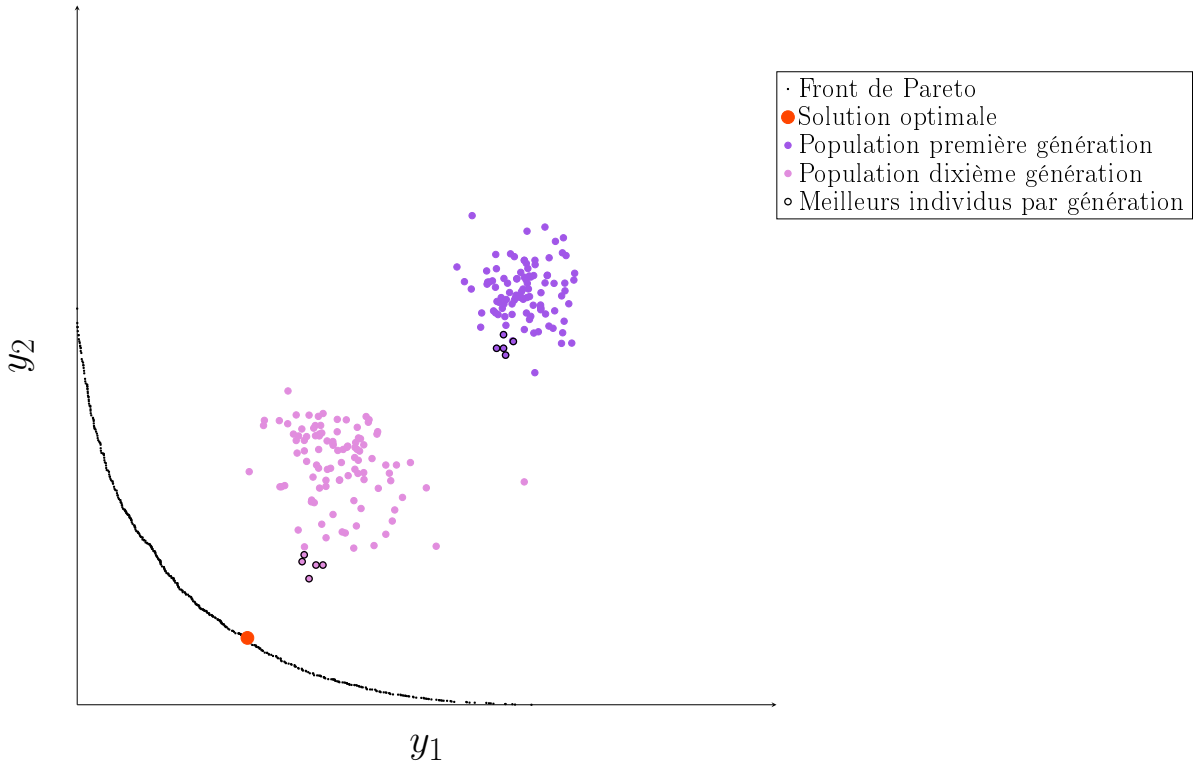


FIGURE 2.2 – Illustration de l'évolution de la population en fonction des générations.

Dans notre contexte, il n'est pas possible d'évaluer directement les solutions d'une population, puisque les préférences du décideur ne sont pas connues initialement. Ainsi, tout comme pour la recherche locale, nous proposons ici de combiner l'approche génétique et l'élicitation incrémentale de préférences, en posant des questions au décideur durant l'évaluation des solutions dans une population. Nous avons testé une version où les opérations de croisement et de mutation sont réalisées sur les solutions du problème, comme ce qui est fait habituellement. Néanmoins, cette version s'est avérée moins performante en pratique que celle qui consiste à croiser et à muter les paramètres de la fonction d'agrégation. Un tableau de résultats viendra appuyer ces affirmations dans les sections correspondant aux applications. Ainsi, nous présentons uniquement la version basée sur les paramètres de la fonction d'agrégation.

Notre algorithme, nommé RIGA pour *Regret-Based Incremental Genetic Algorithm* en anglais, prend en entrée un problème d'optimisation combinatoire multi-objectifs, un seuil de tolérance $\delta \geq 0$, une fonction scalarisante f_λ dont les paramètres λ sont inconnus et un ensemble de données de préférences Θ (potentiellement vide). On commence par générer une population initiale de la manière suivante :

Population initiale : pour générer la population initiale, notée P, nous devons générer un ensemble de paramètres admissibles. Ensuite, pour chaque paramètre généré λ, nous devons déterminer une solution x_λ qui est (presque) optimale pour la fonction de scalarisation f_λ, la population est ainsi consituée de couple (λ, x_λ). Pour ce faire, nous utilisons un algorithme de résolution efficace existant (par exemple, l'algorithme de Prim pour le problème de l'arbre couvrant minimum avec une somme pondérée). Ces paramètres pourraient être générés uniformément au hasard. Cependant, nous proposons plutôt de générer les points extrêmes du polyèdre Λ_Θ, car cela donne de meilleurs résultats en pratique. Notons que, en fonction de l'ensemble initial Θ, le nombre de points extrêmes peut être exponentiel en le nombre d'objectifs (mais ce nombre reste relativement petit si l'ensemble initial Θ est vide). Puis l'algorithme itère les étapes suivantes :

Sélection : à cette étape, nous souhaitons sélectionner les K meilleures couples (λ^k, x^k) de la population P. Pour ce faire, nous allons procéder en plusieurs étapes :

1. On commence par supprimer les solutions Λ_Θ-dominées, c'est-à-dire toutes les solutions x^k pour lesquelles il existe une solution x^l dans la population telle que $f_\lambda(x^l) \leq f_\lambda(x^k)$ pour tout $\lambda \in \Lambda_\Theta$ (avec au moins un $\lambda \in \Lambda$ pour lequel on a $f_\lambda(x^l) < f_\lambda(x^k)$). Cette étape peut être réalisée de manière efficace en pratique en utilisant les sommets du polyèdre Λ_Θ ou la programmation linéaire. On note $\mathcal{X}_P$ l'ensemble des solutions restantes.
2. À présent, on cherche la meilleure solution de l'ensemble $\mathcal{X}_P$. Pour ce faire, on procède comme dans l'algorithme de recherche locale interactive dans les différents voisinages. Plus précisément, tant que $MMR(\mathcal{X}_P, \Lambda_\Theta) > \delta$, on demande au décideur de comparer deux solutions x, x' dans $\mathcal{X}_P$ et on met à jour l'ensemble Λ_Θ en ajoutant la contrainte $f_\lambda(x) \leq f_\lambda(x')$ (ou $f_\lambda(x) \geq f_\lambda(x')$). Une fois que $MMR(\mathcal{X}_P, \Lambda_\Theta) \leq \delta$, on garde une solution x^* arbitrairement choisie dans $\arg\min_{x \in \mathcal{X}_P} MR(x, \mathcal{X}_P, \Lambda_\Theta)$.
3. Enfin, on utilise la solution x^* pour sélectionner les $K - 1$ solutions restantes. Plus

précisément, nous gardons les $K-1$ solutions dans $\mathcal{X}_P$ qui sont les plus proches de x^* au sens de la distance Euclidienne. On note P_K cet ensemble de K solutions.

Croisement : comme mentionné précédemment, les croisements se font sur les vecteurs de paramètres. On utilise un croisement par combinaison linéaire convexe. Plus précisément, à partir de deux couples parents (λ^1, x^1) et (λ^2, x^2) dans la population P_K, on crée un nouveau couple (λ^3, x^3) de la manière suivante. On choisit une valeur $\alpha \in [0,1]$ par tirage aléatoire uniforme, puis on définit $\lambda^3 = \alpha \times \lambda^1 + (1-\alpha) \times \lambda^2$. Cet opérateur de croisement est particulièrement adapté à notre contexte, car à partir de deux paramètres admissibles, on crée un fils dans le polyèdre des paramètres admissibles (puisque le polyèdre est convexe, étant donné que f_λ est linéaire en λ). Si le jeu de poids ainsi formé n'est pas muté, la solution x^3 est ensuite obtenue en résolvant le problème pour la fonction f_{λ^3}. Sinon une mutation, définie de la façon suivante, est appliquée avant la résolution.

Mutation : selon un certain pourcentage de chance, noté μ, une mutation peut être effectuée sur l'individu issu du croisement. Dans notre cas, un nombre aléatoire est tiré selon une loi normale. Dans le but de pouvoir jouer sur l'amplitude de cette valeur, on la multiplie par un certain σ déterminé au préalable. Plus σ est grand, plus la mutation modifiera l'individu. On tire ensuite aléatoirement un critère (un gène) dans le vecteur (génome), λ^3, de l'individu auquel on ajoute la valeur aléatoire multipliée par σ. Après normalisation, le nouvel individu est formé. De la même façon que l'individu non sélectionné pour la mutation, la solution x^3 est ensuite obtenue en résolvant le problème pour la fonction f_{λ^3}

Terminaison : On itère les étapes précédentes jusqu'à ce que le critère d'arrêt soit vérifié. Ici, nous proposons de nous arrêter après un nombre fixé de générations, noté M. L'algorithme retourne alors une solution x^* dans l'ensemble $\arg\min_{x \in P^*} MR(x, \mathcal{X}_P^*, \Lambda_\Theta)$ où P^* est la dernière population.

Cet algorithme est disponible dans une version résumée ci-dessous.

Algorithme 4 `Regret-Based Incremental Genetic Algorithm (RIGA)`

Initialisation des paramètres admissibles : $\Lambda_\Theta \leftarrow \{\lambda : \forall (a,b) \in \Theta, f_\lambda(a) \leq f_\lambda(b)\}$
Génération de la population initiale : $P \leftarrow$ `GenerationPopulationInitiale`(Λ_Θ)
Lancement de la boucle évolutionnaire :
for 1 à M **do**
  $\mathcal{X}_P \leftarrow$ `Croisement&Mutation`$(P, S, \sigma, \mu) \cup P$
  while $MMR(\mathcal{X}_P, \Lambda_\Theta) > \delta$ **do**
    $(x, x') \leftarrow$ `Question`(P^*)
    Mise à jour des données de préférences :
    $\Theta \leftarrow \Theta \cup \{(y(x), y(x'))\}$
    $\Lambda_\Theta \leftarrow \{\lambda : \forall (a,b) \in \Theta, f_\lambda(a) \leq f_\lambda(b)\}$
  end while
  $x^* \leftarrow \arg\min_{x \in P^*} MR(x, \mathcal{X}_P^*, \Lambda_\Theta)$
  $P \leftarrow$ `k-meilleurs`(P, k)
end for
return x^*

L'algorithme 4 prend en entrées un seuil de tolérance δ, μ un taux de mutation et σ un réel qui permet de moduler l'amplitude de la mutation. On a également M, le nombre de générations, et S la taille de la population P. La ligne 2 a pour objectif de déterminer la première population et la boucle 4 à 13 représente la sélection par méthode génétiques

2.1.3 Garanties de performances : temps et nombre de questions

Proposition 2.1.1. *Pour tout problème combinatoire multi-objectifs avec des préférences représentées par un opérateur somme pondérée, un opérateur OWA ou les intégrales de Choquet, si le problème peut être résolu (exactement ou approximativement) en temps polynomial (en la taille du problème) lorsque les préférences sont précisément connues, alors RIGA et ILS peuvent être implémentés de telle sorte qu'ils s'exécutent en temps polynomial et ne demandent pas plus qu'un nombre polynomial de questions.*

En effet, on note que notre algorithme RIGA possède différents paramètres réglables : S la taille de la population (générée par les croisements et les mutations), $K < S$ le nombre de couples sélectionnés pour la génération suivante et M le nombre de pas de la boucle (lignes 4 à 12 dans l'algorithme 4). Notre algorithme ILS possède également un nombre it_{max} d'itérations possibles (ligne 13 dans l'algorithme 3). Afin d'obtenir des méthodes polynomiales, nous devons clairement fixer M, S (pour la méthode génétique) et it_{max} (pour la recherche locale) à certaines valeurs entières qui sont également polynomiales.

Puisque le nombre d'itérations de la boucle externe est polynomial pour ILS et RIGA, le nombre de questions générées est également polynomial si et seulement si le regret minimax de $\mathcal{X}^*$ ($\mathcal{X}_P$ avec RIGA) tombe en dessous de δ après un nombre polynomial de questions de comparaison à chaque itération des procédures de recherche. Pour cela, on peut simplement appliquer la Stratégie de Solution Courante (CSS) [Boutilier et al., 2006] (cf. Section 1.3.2). En procédant ainsi, on peut s'assurer que le regret minimax sera égal à zéro après au plus $|\mathcal{X}^*| - 1$ ($|\mathcal{X}_P| - 1$ avec RIGA) questions puisqu'au moins une solution est éliminée après chaque requête de comparaison.

Nous venons de prouver que le nombre d'itérations est polynomial lors de l'application de la stratégie CSS. Par conséquent, il nous suffit de prouver que les calculs MMR peuvent être effectués en temps polynomial. Rappelons que $MMR(\mathcal{X}^*, \Lambda_\Theta)$ ($MMR(\mathcal{X}_P, \Lambda_\Theta)$ avec RIGA) peut être obtenu en résolvant au plus $|\mathcal{X}^*|^2$ PMR ($|\mathcal{X}_P|^2 = |S|^2$ avec RIGA). Les définitions sont données dans la Section 1.3.1). Par conséquent, il suffit de montrer que les PMR peuvent être calculés en temps polynomial.

On sait que $f_\lambda(a) = \sum_{j=1}^{n} \lambda_j a_j$ est linéaire en a avec λ fixé, et on peut déduire que les calculs des $PMR(x, x', \Lambda_\Theta)$ peuvent être effectués en un temps polynomial avec une somme pondérée.

Pour les agrégateurs OWA, même si $f_\lambda(a) = \sum_{j=1}^{n} \lambda_j a_{(j)}$ est non linéaire en a pour des paramètres fixes λ, il est linéaire en λ pour a fixe. Ainsi, $PMR(x, x', \Lambda_\Theta)$ peut être obtenu en temps polynomial en résolvant le programme linéaire présenté en Section 1.3.1.

Puisque $f_\lambda(a)$ est linéaire en λ pour a fixé avec les intégrales de Choquet, le $PMR(x, x', \Lambda_\Theta)$ peut être obtenu en résolvant le programme linéaire 1.3.2 définit en Section 1.3.1.

2.2 Application au problème du voyageur de commerce

Dans cette section, on présente les résultats numériques obtenus par nos deux méthodes, ILS et RIGA pour le problème du voyageur de commerce symétrique. Dans un premier temps, avant de présenter les performances de nos algorithmes, on présente le problème ainsi que les adaptations nécessaires pour employer ILS ou RIGA.

2.2.1 Formalisation du problème et résolution exacte avec paramètres connus

Le problème du voyageur de commerce symétrique, ou TSP (pour *Traveling Salesman Problem*), est défini par un graphe non orienté $G = (V, R)$, où V est l'ensemble des nœuds et R est l'ensemble des arêtes, et par une fonction de coût associant une évaluation à chaque arête du graphe. Pour le TSP multi-objectifs, la fonction de coût associe un vecteur de valeurs c_r à chaque arête $r \in R$ représentant son évaluation sur les différents objectifs (comme la durée, la distance, le prix, etc.). Pour ce problème, l'ensemble de tous les cycles passant une seule fois par tous les sommets du graphe (aussi appelés cycles hamiltoniens) est l'ensemble $\mathcal{X}$ des solutions réalisables. Ainsi, à toute solution $x \in \mathcal{X}$, on associe le vecteur de performances $y(x)$ défini par la somme des coûts c_r pour toutes les arêtes r appartenant au cycle x.

À titre illustratif, l'exemple suivant décrit une instance de ce problème.

Exemple 2.2.1. *Considérons l'instance du TSP tri-objectifs à 6 villes représentée en Figure 2.3 ; les coûts sont indiqués directement sur les arêtes. Pour ce problème, il y* $\frac{(6-1)!}{2} = 60$ *solutions (cycles hamiltoniens) différentes. Par exemple, le cycle* $x_0 = ABFECDA$ *a pour image le point* $(24, 33, 29)$ *et est représenté en rouge sur la figure.*

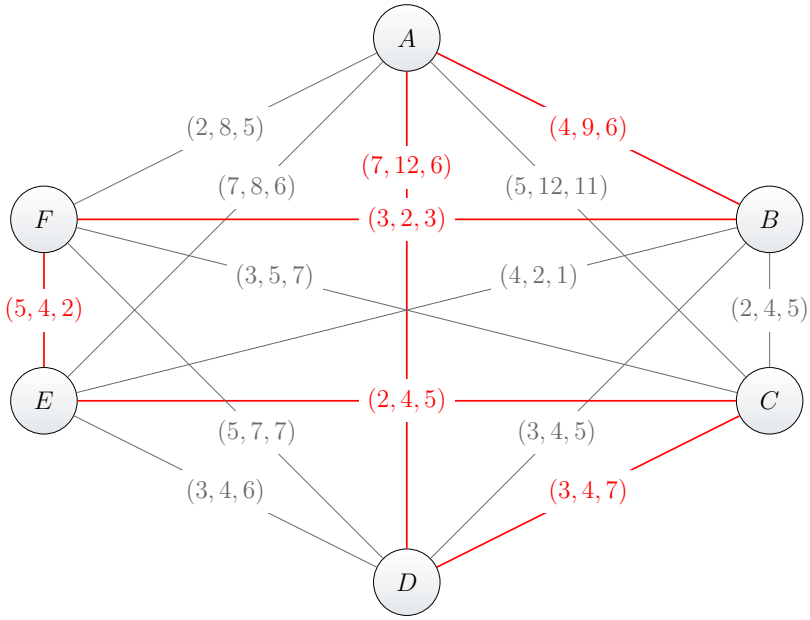


FIGURE 2.3 – Graphe $G = (V, R)$ et $x_0 = ABFECDA$ une solution réalisable.

Résolution exacte avec paramètres connus

Dans le but d'évaluer nos méthodes, nous devons disposer d'un solver dédié à ce problème lorsque les paramètres sont connus. Ainsi, cette sous-section décrit les méthodes utilisées selon la fonction scalararisante choisie pour représenter les préférences du décideur.

Somme pondérée Grâce au caractère linéaire de la somme pondérée, nous pouvons, pour ce problème, utiliser le solveur TSP exact Concorde [1].

Pour OWA et l'intégrale de Choquet, nous utilisons les mêmes méthodes de linéarisation que celles données dans la Section 1.2.1 mais adaptée au TSP symétrique comme présenté ci-dessous.

OWA Dans le but de présenter notre algorithme de résolution exacte, on définit plus formellement le problème du voyageur de commerce en se basant sur la description faite en début de section :

— Données : soit $G = (V, R)$ un graphe complet avec $|V| = q$ villes et c_{ij}^p le vecteur coût correspondant aux scores de l'arête entre la ville i et la ville j. On a $i, j \in \{1, ..., q\}$ et p le nombre de critères.

— Objectif : déterminer un cycle hamiltonien de valeur OWA minimale entre ces q villes.

— Variables : on note $x_{ij} = 1$ si la ville i est suivie de la ville j et 0 sinon.

L'utilisation de l'ingénieuse linéarisation d'un OWA [Ogryczak and Śliwiński, 2003] conduit le problème de maximisation d'un OWA à la formulation du programme linéaire suivant :

$$
\begin{aligned}
&\min \sum_{i=1}^{p} (\lambda_i - \lambda_{i+1})(ir_i + \sum_{j=1}^{p} d_j^i) \\
&s.t.\ r_i + d_j^i \geq y_j(x) && \forall i, j \in \{1, \ldots, p\} \\
&\quad d_j^i \geq 0 && \forall i, j \in \{1, \ldots, p\} \\
&\quad r_i \in \{0, 1\} \\
&\quad d_{ij} \in \{0, 1\}
\end{aligned}
$$

Les termes r et d sont issus de la dualisation du problème principal utilisant la forme de Lorenz pour exprimer le vecteur OWA (cf. Section 1.2.1).

Dans le but de résoudre des instances correspondant au problème du voyageur de commerce de manière efficace, il est nécessaire d'ajouter des contraintes à ce programme. Il est imposé de passer une seule fois par chacune des villes, avec élimination des sous-tours par formulation DFS (pour les auteurs nommés Dantzig, Fulkerson et Johnson [Dantzig et al., 1954]). Nous utilisons une version optimisée de résolution où, pour la première résolution, on impose seulement que chaque sommet soit visité, puis une fois la solution retournée si un, ou des,

1. http://www.math.uwaterloo.ca/tsp/concorde

sous-tours sont formés alors on ajoute une contrainte dans le but de briser ces sous-tours [2]. Ensuite, on réitère avec la ou les, contraintes supplémentaires jusqu'à trouver une solution sans sous-tour. De cette façon, on obtient le programme suivant :

$$\min \sum_{i=1}^{p} (\lambda_i - \lambda_{i+1})(ir_i + \sum_{j=1}^{p} d_j^i)$$

$$s.t.\ r_i + d_j^i \geq y_j(x) \qquad \forall i, j \in \{1, \ldots, p\} \qquad (2.1)$$

$$d_j^i \geq 0 \qquad \forall j, i \in \{1, \ldots, p\} \qquad (2.2)$$

$$\sum_{l \in V} x_{kl} = 1 \qquad \forall k \in \{1, \ldots, |V|\} \qquad (2.3)$$

$$\sum_{k \in V} x_{kl} = 1 \qquad \forall l \in \{1, \ldots, |V|\} \qquad (2.4)$$

Élimination des sous-tours

$$\sum_{i \in S} \sum_{j \in S, i \neq j} x_{ij} \leq |S| - 1 \qquad \forall S \subset V, 2 \leq |S| \leq n - 1 \qquad (2.5)$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in \{1, \ldots, |V|\} \qquad (2.6)$$

Les contraintes (2.1) et (2.2) correspondent à la linéarisation de l'OWA et les contraintes (2.3) et (2.4) correspondent aux contraintes d'incidences, c'est-à-dire imposent deux arêtes incidentes pour chaque sommet. La contrainte (2.5) brise les sous-tours (après vérification) de la solution générée à l'aide des contraintes précédentes. La fonction objectif reste la minimisation d'un OWA.

Intégrale de Choquet 2-additive Avec une intégrale de Choquet, le principe est le même que précédemment, c'est-à-dire que l'on combine les contraintes du problème à la minimisation d'une intégrale de Choquet 2-additive. Ainsi, en utilisant la notion d'inverse de Möbius pour les fonctions de croyance (cf. Section 1.2.2 pour plus de détails), nous obtenons la formulation suivante [Branke et al., 2016] :

$$\min \sum_{A \subseteq \{1, \ldots, p\}} m(A) \min_{j \in A} y_j(x)$$

$$s.t.\ m(\{i\}) \geq 0 \qquad \forall i \in \{1, \ldots, p\} \qquad (2.7)$$

$$m(\{i\}) + \sum_{j \in T} m(\{i, j\}) \geq 0 \qquad \forall i \in \{1, \ldots, p\} \text{ et } \forall T \subseteq \{1, \ldots, p\} \backslash \{i\}, T \neq \emptyset \qquad (2.8)$$

2. Utilisations de LAZYCUT avec CPLEX

C'est une somme pondérée appliquée sur le vecteur à valeurs réelles de taille 2^p dont les composants sont $\min_{j \in A} y_j(x)$ pour $A \subseteq \{1, \ldots, p\}$. La contrainte (2.7) impose que les valeurs des masses de Möbius des singletons soient positives ou nulles. Avec la contrainte (2.8), on contraint la somme de tous les couples formés avec un singleton i à avoir une masse de Möbius inférieur ou égale à la masse de Möbius du singleton i.

À la suite de ces contraintes, il est nécessaire d'ajouter les contraintes correspondantes au problème du voyageur de commerce.

2.2.2 Adaptations d'ILS

Premièrement, un ensemble de $m = 100$ solutions initiales est obtenu en optimisant f_λ à l'aide d'une heuristique, pour 100 vecteurs de pondération générés aléatoirement λ. Pour le modèle de somme pondérée, nous utilisons l'heuristique de Lin-Kernighan [3] qui est l'une des meilleures heuristiques pour résoudre le TSP [Lin and Kernighan, 1973, Applegate et al., 2003] à objectif unique. Pour l'opérateur OWA et l'intégrale de Choquet, nous proposons d'appliquer une recherche locale simple basée sur des échanges 2-opt [Croes, 1958]. En effet, l'utilisation d'une méthode exacte pourrait être prohibitive en termes de temps d'exécution.

Le voisinage 2-opt est l'ensemble des solutions obtenues par échange de deux arêtes. Toutes les combinaisons d'échanges sont considérées. Un exemple d'échange de 2 arêtes est donné en Figure 2.4. On part de la solution $A - C - D - E - B - F - A$ et on obtient la solution $A - D - C - E - B - F - A$

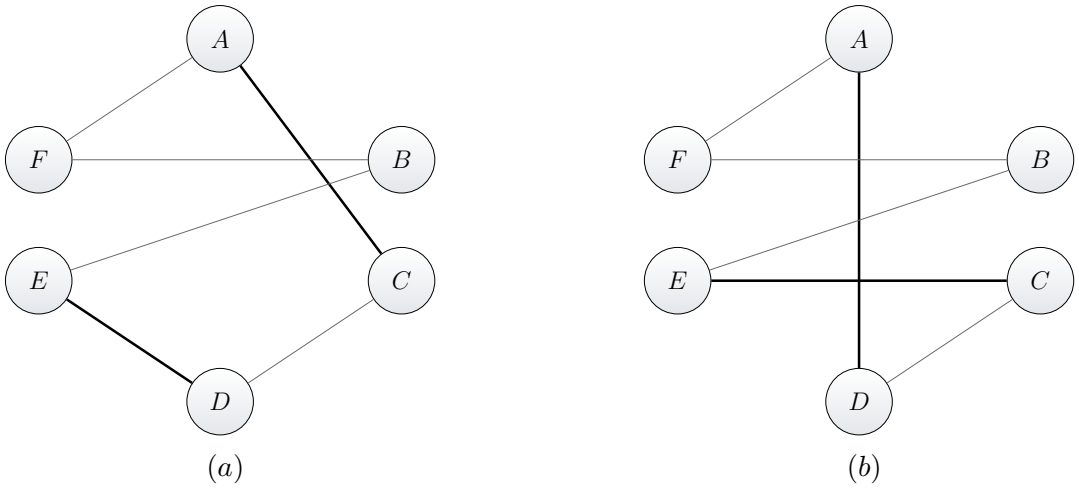


FIGURE 2.4 – Illustration d'un 2-opt.

Sur l'ensemble de solutions initiales ainsi construit, nous sollicitons le décideur jusqu'à ce que le minimax regret tombe en dessous du seuil $\delta_1 = 0.1$ (soit 10% du regret initial), et ensuite la solution de départ est choisie arbitrairement parmi les solutions optimales selon le critère de décision du regret minimax.

3. http://www.math.uwaterloo.ca/tsp/concorde

Pour la procédure de recherche locale, la fonction de voisinage est également définie par un algorithme 2-opt. Afin de sélectionner la solution courante suivante, on pose des questions pour discriminer les solutions non dominées au sens de Pareto au sein du voisinage. Des tests ont été réalisés avec un voisinage 3-opt (échange de 3 arêtes). Le temps d'exécution ainsi que le nombre de questions ont considérablement augmenté alors que la qualité de la solution retournée ne se trouvait pas améliorer de façon notable. En effet, les tailles de voisinages sont plus importantes avec l'utilisation d'un 3-opt.

À chaque étape de l'itération, nous posons des questions entre deux solutions de ce voisinage jusqu'à ce que le regret minimax tombe en dessous d'un seuil donné $\delta_2 = 0.4$ puis nous nous déplaçons vers une solution voisine qui minimise le regret maximal.

Les seuils δ_1 et δ_2 utilisés pour ces méthodes sont issus d'une série de tests permettant d'identifier les paramètres donnant les meilleurs résultats (c'est-à-dire un bon compromis entre nombre de questions, temps d'exécution et qualité de la solution). En effet, si $\delta_1 = 0$ alors on pose trop de questions avant le lancement de la recherche locale, ce qui pénalise la méthode. A l'inverse, si $\delta_1 \geq 0.2$ alors la solution initiale n'est pas de qualité suffisante et la recherche locale effectue plus d'itérations (temps d'exécution plus lent et nombre de questions plus élevé). De la même façon, si δ_2 est trop petit ($\delta_2 \leq 0.3$) alors on pose trop de questions mais s'il est trop grand ($\delta_2 \geq 0.5$) alors la recherche locale s'arrête trop tôt et la solution est de moins bonne qualité. La valeur m (nombre de points de départ) est aussi déterminée par des tests, ainsi, si m est trop petit ($m = 50$), alors la qualité de la solution initiale est trop faible, et si m est trop grand ($m \geq 150$) alors on pose trop de questions afin de déterminer la meilleure solution de départ.

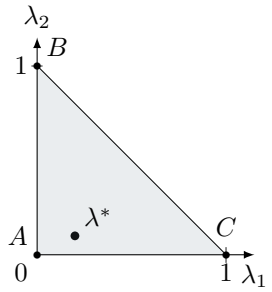
La section suivante présente une exécution de notre algorithme.

2.2.3 Illustration de l'algorithme de recherche locale interactif

Dans l'exemple suivant, les paramètres employés sont simplifiés. Ainsi, on applique l'algorithme ILS avec $\delta = (\delta_1, \delta_2) = (0, 0)$ sur une petite instance en considérant un voisinage 2-opt. Seulement deux solutions de départ potentielles sont utilisées.

Exemple 2.2.2. *Soit l'instance du TSP tri-objectifs de* 6 *villes présenté en Figure 2.3. L'ensemble $\mathcal{X}$ de solutions réalisables est l'ensemble de tous les cycles Hamiltoniens.*

Les préférences du décideur sont représentées par une somme pondérée définie avec le jeu de poids caché $\lambda^* = (0.2, 0.1, 0.7)$ et nous commençons l'exécution avec un ensemble vide de données de préférences connues, $\Theta = \emptyset$. Ainsi, Λ_Θ est initialement l'ensemble de tous les vecteurs de paramètres possibles, $\lambda = (\lambda_1, \lambda_2, \lambda_3) \in [0, 1]^3$ tels que $\lambda_1 + \lambda_2 + \lambda_3 = 1$. On représente Λ_Θ par le triangle en Figure 2.5 (dans l'espace formé par (λ_1, λ_2), λ_3 est implicitement défini par $\lambda_3 = 1 - \lambda_1 - \lambda_2$).



(*a*)

FIGURE 2.5 – Λ_Θ intial.

Identification d'une bonne solution de départ : *Dans un premier temps, on choisit de générer* $m = 2$ *vecteurs poids* λ^1 *et* λ^2 *de manière aléatoire et on détermine alors les solutions optimales correspondantes* x^1 *and* x^2*. Si* $\lambda^1 = (0.6, 0.3, 0.1)$ *et* $\lambda^2 = (0.3, 0.6, 0.1)$*, nous obtenons les cycles Hamiltoniens* x^1 *et* x^2 *avec l'évaluation suivante :* $y(x^1) = (19, 34, 30)$ *et* $y(x^2) = (21, 32, 27)$*. Ainsi* $\mathcal{X}_0 = \{x^1, x^2\}$*. Puisque* $MMR(\mathcal{X}_0, \Lambda_\Theta) = 2 > \delta_1$*, on demande au décideur de comparer* x^1 *et* x^2*. Comme* $f_{\lambda^*}(y(x^1)) = 28.2 > f_{\lambda^*}(y(x^2)) = 26.3$*, le décideur préfère la solution* x^2 *à* x^1*. Ainsi, on met-à-jour l'ensemble* $\Theta = \{((21, 32, 27), (19, 34, 30))\}$ *et* Λ_Θ *est restreint en imposant la contrainte* $f_\lambda(y(x^2)) \leq f_\lambda(y(x^1))$*, c'est-à-dire* $\lambda_2 \leq -5\lambda_1 + 3$*. On peut observer cela en Figure 2.6*(*a*) *où* Λ_Θ *est représenté par ABDE. Maintenant, on a* $MMR(\mathcal{X}_0, \Lambda_\Theta) = MR(x^2, \mathcal{X}_0, \Lambda_\Theta) = 0 \leq \delta_1$*, et donc* x^2 *est choisie pour être la solution de départ (c'est-à-dire* $x^* = x^2$*).*

Recherche Locale : *À la première étape, trois voisins 2-opt de* x^* *sont Pareto non-dominés, et l'ensemble* $\mathcal{X}^*$ *possède quatre solutions, notées* x^1*,* $x^2 (= x^*)$*,* x^3 *et* x^4 *évaluées de la manière suivante :* $y(x^1) = (23, 34, 26)$*,* $y(x^2) = (21, 32, 27)$*,* $y(x^3) = (19, 34, 30)$ *et* $y(x^4) = (20, 31, 30)$*. Puisque* $MMR(\mathcal{X}^*, \Lambda_\Theta) = 1 > \delta_2$*, on demande au décideur de comparer deux solutions dans* $\mathcal{X}^*$ *:* x^1 *et* x^**. Comme* $f_{\lambda^*}(y(x^1)) = 26.2 < f_{\lambda^*}(y(x^*)) = 26.3$*, le décideur préfère* x^1 *à* x^**. Nous obtenons alors* $\Theta = \{((21, 32, 27), (19, 34, 30)), ((23, 34, 26), (21, 32, 27))\}$ *et* Λ_Θ *est réduit par la contrainte linéaire* $f_\lambda(y(x^1)) \leq f_\lambda(y(x^*))$*, c'est-à-dire* $\lambda_2 \leq -\lambda_1 + \frac{1}{3}\lambda_3$*. Nous pouvons l'observer en Figure 2.6*(*b*) *où* Λ_Θ *est représenté par le triangle AGF. Nous arrêtons alors de poser des questions à cette étape puisque nous avons* $MMR(\mathcal{X}^*, \Lambda_\Theta) = MR(x^1, \mathcal{X}^*, \Lambda_\Theta) = 0 \leq \delta_2$*. On se déplace de* $x^* = x^2$ *à* x^1 *pour la prochaine étape (c'est-à-dire* $x^* = x^1$*).*

À l'itération suivante, $\mathcal{X}^*$ *inclut seulement trois solutions non dominées, notées* $x^1 (= x^*)$*,* x^2 *et* x^3 *avec* $y(x^1) = (23, 34, 26)$*,* $y(x^2) = (21, 32, 27)$ *et* $y(x^3) = (19, 33, 31)$*. Puisque* $MMR(\mathcal{X}^*, \Lambda_\Theta) = 0 \leq \delta_2$*, aucune question n'est générée à cette étape. De plus,* $MR(x^*, \mathcal{X}^*, \Lambda_\Theta) = 0 \leq \delta_2$ *et* x^* *est donc un optimum local (la variable amélioration passe à false et la boucle s'arrête).*

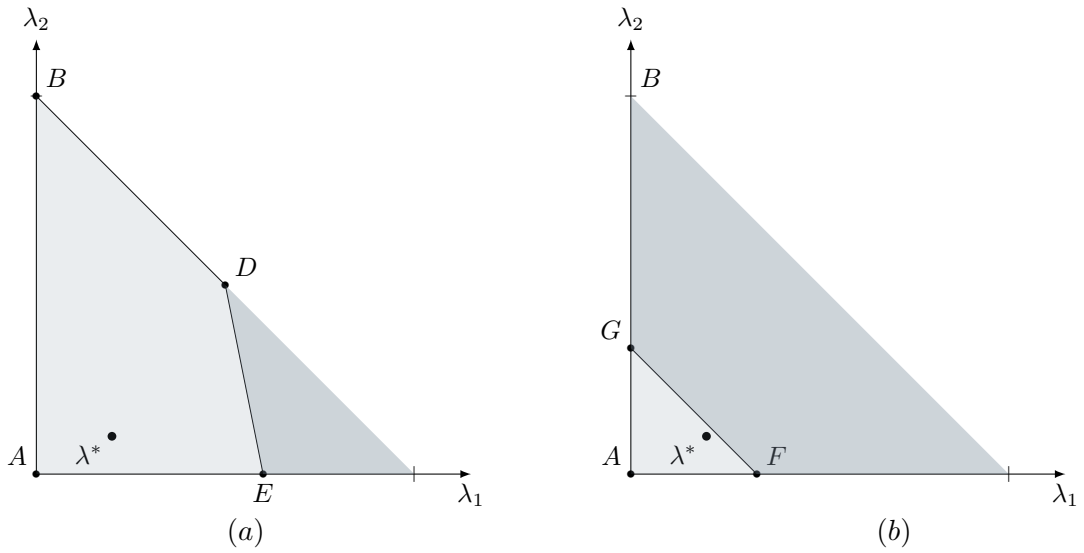


FIGURE 2.6 – Évolution de Λ_Θ au cours de l'exécution de l'algorithme 3.

Après deux itérations, l'algorithme ILS s'arrête donc et retourne la solution $x^* = x^1$ *d'évaluation* $(23, 34, 26)$ *et correspondant au cycle* $A - D - C - B - E - F - A$ *qui est la solution optimale du décideur pour ses paramètres et ce problème. On remarque que seulement deux questions ont été nécessaires pour la déterminer parmi* 60 *solutions réalisables,dont 10 sont Pareto-Optimales.*

Dans la section suivante, on s'intéresse à notre algorithme RIGA basé sur le principe de la génétique et sur les adaptations faites pour son application au problème du voyageur de commerce.

2.2.4 Adaptations de RIGA

Rappelons que RIGA génère de nouvelles solutions en utilisant une méthode de résolution existante conçue pour le problème avec des poids connus. Pareillement à la recherche locale, la résolution exacte du problème du voyageur de commerce multi-objectifs avec poids précis peut être prohibitive en termes de temps d'exécution. Par conséquent, nous proposons plutôt d'utiliser des méthodes heuristiques pour obtenir des solutions de bonne qualité plus efficacement. Pour le modèle de somme pondérée, nous utilisons l'heuristique de Lin-Kernighan [4] et pour les intégrales OWA et de Choquet, on utilise la recherche locale simple avec le voisinage 2-opt. RIGA implémenté avec ces heuristiques sera désigné par RIGA_H par la suite, tandis que RIGA utilisant des méthodes exactes est simplement noté RIGA.

Le Tableau 2.1 montre les résultats obtenus par RIGA et RIGA_H sur les mêmes instances euclidiennes symétriques du TSP avec 50 villes. Comme prévu, RIGA est très lent et le délai autorisé (de 900 secondes) est dépassé lorsqu'on considère 5 critères ou plus pour OWA et Choquet. Nous observons également que le temps de calcul est considérablement réduit

4. http://www.math.uwaterloo.ca/tsp/concorde

		Somme pondérée			OWA			Choquet		
méthode	p	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)
RIGA	3	66.17	14.8	0.00	546.86	5.5	0.00	235.63	23.3	0.60
	5	68.35	22.6	0.16	/	/	/	/	/	/
RIGA_H	3	11.57	14.9	0.01	5.80	4.1	0.91	15.58	23.6	0.69
	5	10.45	22.2	0.10	3.75	2.7	0.46	16.67	27.9	0.80

TABLE 2.1 – Résultats obtenus par RIGA et RIGA_H avec $\delta = 0$, $M = 20$, $S = 40$ and $K = 5$.

lorsque l'on utilise des heuristiques au lieu de méthodes exactes. Par exemple, pour des instances avec 3 critères, RIGA_H est 6 fois plus rapide que RIGA pour la somme pondérée, 109 fois plus rapide pour OWA et 15 fois pour Choquet. De plus, les erreurs obtenues par RIGA_H ne sont que légèrement supérieures à celles de RIGA, et le nombre de questions est presque identique. Par conséquent, RIGA_H surpasse clairement RIGA pour le TSP et c'est cette adaptation que nous utiliserons dans la suite de cette section.

Comme constaté lors de la description des méthodes évolutionnaires générales (cf. Section 1.1.3) ou encore lors de la description détaillée de notre algorithme RIGA, de nombreux paramètres sont à définir pour ce type de méthode. Afin de déterminer les meilleures valeurs pour RIGA_H, nous exécutons l'algorithme en considérant toutes les combinaisons possibles entre ces différents paramètres : $M = 10, 20, 30$, $S = 20, 40, 60$, et $K = 2, 5, 10$ (avec M le nombre de générations, S la taille de la population et K la taille de la sélection). Les résultats numériques de ces tests sont disponibles dans le Tableau 2.2.

			Somme pondérée			**OWA**			**Choquet**		
M	S	K	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)
10	20	2	4.85	12.9	1.34	0.98	2.2	0.49	5.26	18.0	2.59
		5	5.31	12.8	1.40	1.32	2.3	0.50	5.12	18.4	3.10
		10	5.76	13.1	1.53	1.13	2.4	0.51	4.89	17.6	3.00
	40	2	5.48	13.0	1.81	1.95	2.2	0.46	10.74	18.2	1.93
		5	5.88	12.9	3.52	2.00	2.7	0.47	10.47	18.0	1.70
		10	6.32	13.1	3.22	2.12	2.6	0.48	9.97	17.8	1.87
	60	2	7.57	12.8	1.10	2.38	2.4	0.46	14.45	17.6	1.44
		5	7.94	12.9	1.20	2.80	3.1	0.46	15.44	18.5	1.72
		10	9.12	12.8	1.54	2.77	2.9	0.46	13.81	17.1	1.63
20	20	2	5.31	22.3	0.22	2.05	2.3	0.47	9.56	27.8	1.10
		5	5.81	22.2	0.29	2.00	2.7	0.48	10.20	28.5	1.17
		10	7.12	22.3	0.43	2.28	2.8	0.48	8.94	27.3	1.28
	40	2	9.68	22.4	0.13	3.23	2.5	0.46	15.48	23.7	0.67
		5	10.45	22.2	0.10	3.75	2.7	0.46	16.64	27.9	0.80
		10	12.96	22.8	0.22	3.82	3.4	0.47	17.39	28.4	1.03
	60	2	14.84	22.6	0.14	5.01	2.8	0.46	25.37	28.1	0.92
		5	15.34	22.5	0.14	4.95	3.9	0.46	25.62	27.5	0.82
		10	15.79	22.7	0.07	5.45	4.1	0.46	26.01	27.8	1.01
30	20	2	7.34	28.2	0.04	2.68	2.6	0.46	12.15	38.4	0.83
		5	7.98	28.1	0.06	3.02	2.9	0.47	12.08	38.4	0.82
		10	9.60	29.7	0.21	3.50	2.7	0.46	12.40	37.7	0.92
	40	2	13.48	28.8	0.15	4.92	3.2	0.46	23.13	37.7	0.58
		5	14.98	29.0	0.06	5.26	3.9	0.46	23.99	37.9	0.66
		10	17.50	30.3	0.06	6.03	3.9	0.46	25.75	37.5	0.66
	60	2	21.23	29.4	0.02	8.62	2.5	0.46	36.05	38.3	0.51
		5	22.32	29.5	0.08	7.79	4.1	0.46	36.19	37.5	0.61
		10	24.02	30.2	0.08	8.68	4.9	0.46	37.04	38.1	0.65

TABLE 2.2 – Résultats obtenus avec variation des paramètres M, S et K par RIGA_H avec $p = 5$, $\delta = 0.5$, et $\mu = 0.5$.

On voit que fixer $M = 20$, $S = 40$, et $K = 5$ donne le meilleur compromis en termes de temps de calcul, de nombre de questions et d'erreur. En particulier, l'erreur devient trop élevée lorsque le nombre de générations est faible (c'est-à-dire $M = 10$), alors que le nombre de questions et les temps de calcul deviennent trop importants lorsque le nombre de générations est grand (c'est-à-dire $M = 30$). L'algorithme RIGA$_H$ est trop lent lorsqu'il considère $S = 60$, sans réduire l'erreur de manière significative. Plus la taille de la sélection (K) augmente, et plus le nombre de questions augmente, sans que la distance à l'optimale soit nécessairement inférieure. Ainsi c'est la valeur compromis, soit $K = 5$, qui a été retenue.

Maintenant, nous cherchons à évaluer l'impact du taux de mutation sur les performances de notre algorithme. Pour ce faire, nous utilisons différents taux de mutation : $\mu = 0.2, 0.5, 0.8$. Le paramètre $\mu = 0.5$ conduit à un bon compromis entre le nombre de questions, le temps de calcul et l'erreur. En particulier, l'erreur augmente en considérant une valeur plus petite (c'est-à-dire $\mu = 0.2$) car elle ne permet pas une diversité suffisante. Le temps de calcul et l'erreur augmentent en utilisant une valeur plus grande (c'est-à-dire $\mu = 0.8$). Nous utiliserons donc $\mu = 0.5$ par la suite.

	Somme pondérée			OWA			Choquet		
μ	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)
0.2	11.16	22.4	0.59	3.82	2.5	0.50	15.12	26.9	1.48
0.5	10.45	22.2	0.10	3.75	2.7	0.46	16.64	27.9	0.80
0.8	12.76	22.6	0.14	3.98	2.8	0.48	18.03	27.6	1.00

TABLE 2.3 – Résultats obtenus avec $\mu = \{0.2, 0.5, 0.8\}$ par RIGA$_H$ avec $p = 5$.

Pareillement, différents seuils de tolérance ont été considérés et la valeur $\delta = 0.5$ conduit à un bon compromis entre le nombre de questions, le temps de calcul et l'erreur. En effet, l'utilisation d'une valeur plus petite induit un plus grand nombre de questions car plus de questions sont nécessaires afin de descendre en dessous de la valeur de regret visée. Notez que l'augmentation de δ réduit les temps de calcul pour la somme pondérée et Choquet, mais pas pour OWA. Cela est dû au fait que la méthode utilisée pour résoudre le problème avec des poids connus est moins efficace lorsqu'on considère des poids non équilibrés (c'est-à-dire très différents de $(\frac{1}{p}, \ldots, \frac{1}{p})$), qui sont plus susceptibles d'être sélectionnés lorsqu'on considère un seuil de tolérance plus grand.

Enfin, nous avons implémenté trois versions de notre algorithme :

— RIGA : l'algorithme proposé.

— RIGA$_{KCSS}$: l'algorithme proposé mais au lieu d'utiliser la distance euclidienne pour sélectionner les solutions à chaque étape de l'itération, nous sélectionnons les solutions une par une en utilisant la stratégie de génération de questions CSS. Plus précisément, nous générons d'abord des questions de préférences selon la stratégie CSS, jusqu'à identifier une solution dont le regret maximal est inférieur au seuil de tolérance donné. Ensuite, cette solution est retirée de la population et le processus de sélection est itéré sur l'ensemble des solutions restantes avec le même seuil de tolérance. Ce processus s'arrête après avoir sélectionné K solutions.

— RIGA$_S$: l'algorithme proposé mais au lieu d'appliquer les opérateurs génétiques sur les vecteurs de paramètres, ils sont directement appliqués sur les solutions. Nous utili-

sons ici des croisements en deux points et des mutations par permutation (les mêmes opérateurs génétiques que ceux de NEMO, méthode développée dans le Chapitre 1).

En pratique RIGA$_S$ renvoie des solutions de très faible qualité, l'erreur étant supérieure à 20% même en considérant uniquement 3 critères. Pour réduire son erreur, il faut augmenter le nombre de générations, mais la méthode pose alors trop de questions pour atteindre le même niveau de qualité que RIGA$_H$. Pour donner un exemple, pour $p = 6$ critères, nous avons réussi à obtenir une erreur inférieure à 2% avec $M = 1000$ générations, $S = 300$ et $K = 30$ mais une moyenne de 75 questions ont été générées pendant l'exécution. De même, c'est logiquement que RIGA$_H$ surpasse RIGA$_{KCSS}$ implémenté avec des heuristiques. L'application des opérateurs génétiques sur les vecteurs de paramètres est plus efficace que leur application sur les solutions réalisables. Notez également que RIGA$_S$ est beaucoup plus rapide que RIGA, car il ne nécessite pas de résoudre le problème considéré avec des poids précis à chaque itération.

La section suivante présente une exécution de RIGA sur une petite instance du voyageur de commerce.

2.2.5 Illustration de l'algorithme génétique interactif

Dans l'exemple ci-dessous et de la même manière que pour ILS, on illustre l'algorithme RIGA avec des paramètres simplifiés. Ainsi, nous appliquons RIGA sur l'instance suivante avec $\delta = 0$, $S = 5$, $K = 2$, $M = 2$. Le taux de mutation reste inchangé, soit $\mu = 0.5$.

Exemple 2.2.3. *L'instance du problème multi-objectif du voyageur de commerce est la même que celle de l'exemple précédent (cf. Figure 2.3). L'instance est composée de* 6 *villes avec* $p = 3$. *Nous supposons ici que les préférences du décideur peuvent être représentées par un opérateur OWA avec le poids croissant caché de la procédure* $\lambda^* = (0.1, 0.3, 0.6)$.

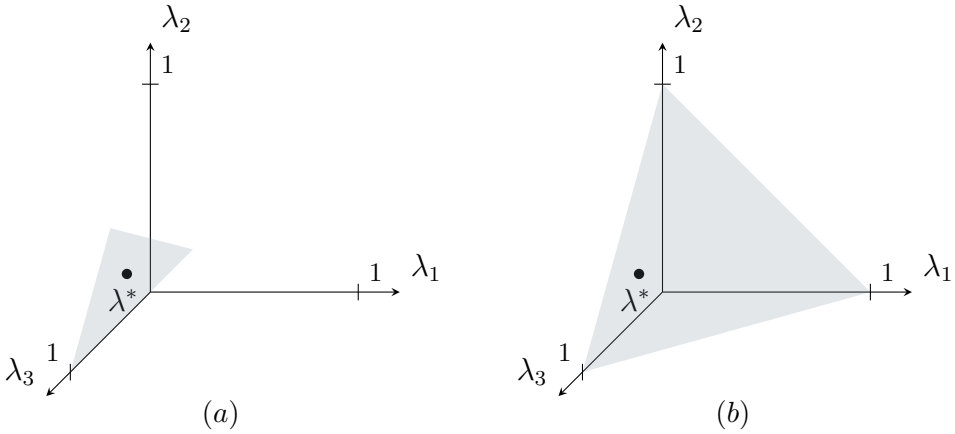


FIGURE 2.7 – Représentation de Λ_Θ selon l'utilisation d'un OWA à poids croissants (a) ou d'une somme pondérée (b).

En comparaison à la somme pondérée, un OWA équitable réduit considérablement la taille de l'ensemble des paramètres admissibles, comme nous pouvons l'observer en Figure 2.7.

Phase d'initialisation : *L'ensemble des vecteurs poids admissibles est initialement défini par* $\Lambda_\Theta = \{\lambda = (\lambda_1, \lambda_2, \lambda_3) \in [0,1]^3 : \lambda_1 + \lambda_2 + \lambda_3 = 1 \text{ et } \lambda_1 \leq \lambda_2 \leq \lambda_3\}$. *Notez que nous pouvons supposer que* Λ_Θ *est un polyèdre convexe tout au long de cette sous-section puisque toute contrainte du type* $f_\lambda(a) \leq f_\lambda(b)$ *est linéaire en* λ *pour tout vecteur de performance fixe* a. *Dans la Figure 2.8, l'ensemble initial* Λ_Θ *est représenté par le triangle dans l'espace* (λ_1, λ_2), λ_3 *étant défini par* $\lambda_3 = 1 - \lambda_1 - \lambda_2$ *(pour plus de lisibilité dû à l'utilisation d'un OWA équitable). Les points extrêmes de* Λ_Θ *sont* $(0, 0.5, 0.5)$, $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ *et* $(0, 0, 1)$.

Population initiale : *Afin de générer la population initiale, nous déterminons une solution quasi-optimale pour chaque point extrême du polyèdre* Λ_Θ *(en utilisant une procédure de recherche locale par exemple). On obtient* $P = \{(\lambda^A, x^A), (\lambda^B, x^B), (\lambda^C, x^C)\}$ *où* $\lambda^A = (0, 0.5, 0.5)$, $\lambda^B = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, $\lambda^C = (0, 0, 1)$, $y(x^A) = (49, 52, 60)$, $y(x^B) = (39, 50, 66)$, *et* $y(x^C) = (56, 57, 58)$.

Première itération : *Puisque* $|P| = 3$ *et* $S = 5$, *nous devons générer* 2 *individus supplémentaires. En appliquant l'opérateur de croisement sur* $\lambda^A = (0, 0.5, 0.5)$ *et* $\lambda^B = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, *puis en effectuant une mutation gaussienne sur le premier objectif, on obtient le vecteur suivant* $\lambda^1 = (0.27, 0.33, 0.40)$. *La fonction* f_{λ^1} *est alors optimisée, ce qui permet de générer la solution* x^1 *dont le vecteur de coût est* $(39, 50, 66)$. *Le couple* (λ^1, x^1) *est ensuite ajouté à* P. *En appliquant l'opérateur de croisement sur* $\lambda^B = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ *et* $\lambda^C = (0, 0, 1)$, *et après avoir effectué une mutation gaussienne sur le second objectif, on obtient* $\lambda^2 = (0.27, 0.33, 0.40)$ *et le cycle* x^2 *dont le vecteur de coût est* $(56, 57, 58)$. *Le couple* (λ^2, x^2) *est alors inséré dans* P. *On a maintenant une population complète.*

L'étape de sélection commence. Nous posons des questions au décideur jusqu'à ce que $MMR(X_P, \Lambda_\Theta) \leq \delta = 0$, *où* $X_P = (x^A, x^B, x^C, x^1, x^2)$. *Puisque* $MMR(X_P, \Lambda_\Theta) = 2 > 0$, *nous demandons au décideur de comparer deux solutions dans* X_P, *disons* x^A *et* x^B. *Puisque* $f_{\lambda^*}(y(x^A)) = 56.5 < f_{\lambda^*}(y(x^B)) = 58.5$, *le décideur répond : "La solution* x^A *est meilleure que la solution* x^B*". Alors* Θ *est mis à jour en ajoutant le couple* $(y(x^A), y(x^B))$ *; on a donc* $\Theta = \{((49, 52, 60), (39, 50, 66))\}$. *Par conséquent, l'ensemble des paramètres admissibles* Λ_Θ *est restreint par la contrainte linéaire* $f_\lambda(y(x^A)) \leq f_\lambda(y(x^B))$, *c'est-à-dire* $\lambda_2 \leq 3\lambda_3 - 5\lambda_1$. *Maintenant,* Λ_Θ *est représenté par le triangle DCE dans la Figure 2.8*(a), *et nous avons* $MMR(X_P, \Lambda_\Theta) = 2 > 0$. *Puisque le regret minimax est supérieur au seuil, nous demandons au décideur de comparer deux solutions dans* X_P, *disons* x^C *et* x^B. *Le décideur préfère la solution* x^B *à la solution* x^C *puisque nous avons* $f_{\lambda^*}(y(x^B)) = 56.5 < f_{\lambda^*}(y(x^C)) = 57.5$. *On met alors à jour* Θ *en insérant le couple* $((49, 52, 60), (56, 57, 58))$ *et on restreint* Λ_Θ *en imposant la contrainte linéaire* $f_\lambda(y(x^A)) \leq f_\lambda(y(x^C))$, *c'est-à-dire que* $\lambda_2 \leq \frac{2}{5}\lambda_3 - \frac{7}{5}\lambda_1$ *(*Λ_Θ *est maintenant représenté par DCFG dans la Figure 2.8*(b)*). Nous avons maintenant* $MMR(X_P, \Lambda_\Theta) = MR(x^A, X_P, \Lambda_\Theta) = 0$ *(la boucle while s'arrête). Nous devons sélectionner des solutions pour la population suivante. Ici, nous avons* $x^* = x^A$. *Puisque* $K = 2$, *nous devons sélectionner un couple supplémentaire pour la prochaine génération. Nous choisissons* x^C *car c'est la solution la plus proche de* x^* *selon la distance euclidienne. Ainsi, nous avons* $P = \{(\lambda^C, x^C), (\lambda^A, x^A)\}$ *pour la prochaine étape.*

Seconde itération : *Puisque* $|P| = 2$ *et* $S = 5$, *nous devons générer trois couples supplémentaires. Après avoir appliqué les opérateurs de croisement et de mutation sur* λ^A *et*

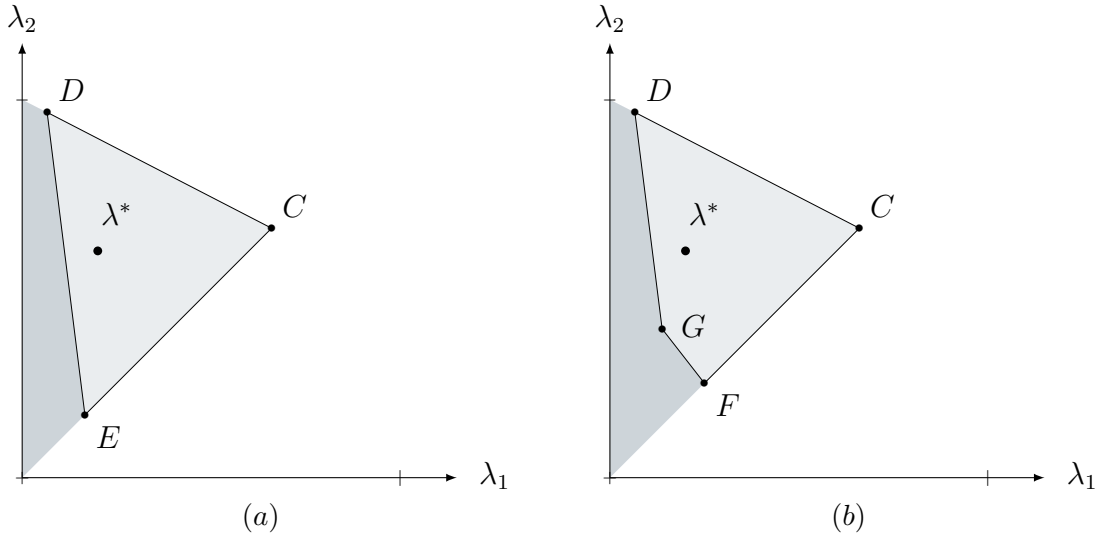


FIGURE 2.8 – Évolution de Λ_Θ durant l'exécution de l'algorithme 4.

λ^C, nous obtenons $(0, 0.41, 0.59)$, $(0, 0.21, 0.79)$ *et* $(0, 0.18, 0.82)$. *Nous optimisons ensuite les fonctions OWA correspondantes et nous obtenons des solutions* x^3, x^4 *et* x^5 *dont les vecteurs de coût sont* $y(x^3) = (49, 52, 60)$, $y(x^4) = (56, 57, 58)$ *et* $y(x^5) = (56, 57, 58)$ *respectivement. Par conséquent, nous avons* $P = \{(\lambda^A, x^A), (\lambda^C, x^C), (\lambda^3, x^3), (\lambda^4, x^4), (\lambda^5, x^5)\}$. *Nous n'avons pas besoin de poser une question à cette étape car* $MMR(X_P, \Lambda_\Theta) = MR(x^A, X_P, \Lambda_\Theta) = 0 \leq \delta$. *À cette étape, nous avons* $x^* = x^A$.

Puisque $M = 2$, *RIGA n'effectue que deux d'itérations et s'arrête en retournant la solution* $x^* = x^A$ *(correspondant au cycle* $A - D - C - B - E - F - A$*), qui est ici optimale selon les préférences du décideur. Dans cet exemple, seulement* 2 *questions ont été nécessaires pour discriminer les* 10 *solutions Pareto-optimales.*

2.2.6 Résultats expérimentaux

Nous comparons les performances de nos algorithmes à celles obtenues par les méthodes existantes suivantes :

Necessary-preference-enhanced Evolutionary Multi-objective Optimizer (NEMO) Dans cet algorithme génétique [Branke and Deb, 2005] (décrit plus en détail en Section 1.1.3), les opérateurs de mutation et de croisement sont appliqués sur les solutions (au lieu des vecteurs de pondération), et une méthode de sélection par tournoi est utilisée pour construire les populations. L'opérateur de mutation consiste simplement à échanger la position de deux gènes choisis au hasard. L'opérateur de croisement dépend du problème considéré : nous utilisons ici le croisement en deux points. Nous effectuons une recherche locale pour chaque solution générée afin d'obtenir une population de meilleure qualité. Plus précisément, nous partons de la solution générée, et nous utilisons une recherche locale simple basée sur la fonction de voisinage 2-opt. Les améliorations locales sont définies

par une fonction d'agrégation donnée f_λ, générée aléatoirement avant l'exécution de la recherche locale.

À chaque étape de l'itération, la programmation linéaire est utilisée pour classer les solutions dans la population actuelle, en utilisant les informations de préférence recueillies ainsi que la crowding distance. Dans cette méthode, une requête de préférence est générée toutes les 10 générations : on demande au décideur de comparer deux solutions potentiellement bonnes sélectionnées parmi celles de la population actuelle. Dans nos expériences, la taille de la population est fixée à 30 et le taux de mutation est fixé à $\frac{1}{p}$, comme proposé dans l'article initial [Branke and Deb, 2005]. De plus, le nombre de solutions sélectionnées pour construire la génération suivante est fixé à 5 à chaque étape de l'itération. En effet, les meilleurs résultats ont été obtenus avec ces valeurs de paramètres, car tout comme RIGA, le même soin a été apporté à la détermination des valeurs des paramètres de NEMO.

Méthode en deux phases (Two-Phase) Cette méthode consiste à construire dans un premier temps l'ensemble, ou une approximation de l'ensemble, des solutions efficaces (ensemble de Pareto), puis à appliquer la stratégie CSS sur cet ensemble jusqu'à ce que le regret minimax tombe sous le seuil $\delta \geq 0$.

Afin de générer efficacement l'ensemble des points efficaces pour ce problème, une approximation de bonne qualité est faite en utilisant la méthode heuristique proposée dans [Jaszkiewicz, 2018] (basée sur la recherche locale). Nous sélectionnons ensuite aléatoirement 3000 solutions, afin de réduire les temps de calcul de la stratégie CSS, prohibitif sinon. Notez que les temps de calcul que nous donneront ne comprennent pas le temps nécessaire pour générer l'approximation de l'ensemble de Pareto.

Incremental Elicitation based on Extreme Points (IEEP) [Benabbou and Lust, 2019a] La méthode de résolution exacte introduite ici est basée sur les points extrêmes du polyèdre représentant les paramètres de préférence admissibles Λ_Θ (cf. Section 1.3.3). Pour chaque point extrême, une méthode de résolution exacte est utilisée pour obtenir la solution optimale correspondante. Des questions sont aussi posées au décideur afin de déterminer la meilleure des solutions correspondant à un des poids extrêmes, et ainsi de suite jusqu'à convergence. Cette méthode interactive garantit de retourner une solution avec un regret maximum inférieur à un seuil donné $\delta \geq 0$ à la fin de l'exécution, à condition que des méthodes de résolution exactes soient utilisées (et non des heuristiques).

Résultats numériques

Nous présentons maintenant les résultats obtenus. Tous les algorithmes (ainsi que ceux présentés dans la suite de cette thèse) sont codés en C++ et testés sur un Intel Core i7-9700, 3.00 GHz avec 15,5 GB de RAM. Les optimisations des PMR ont été effectuées par CPLEX [5].

5. https://www.ibm.com/analytics/cplex-optimizer

Nous récupérons, à l'aide du logiciel Polymake [6], les points extrêmes du polyèdre des jeux de poids possibles (utilisé également pour l'algorithme IEEP).

Nous considérons des instances euclidiennes symétriques [7] du TSP avec 50 villes et un nombre d'objectifs p appartenant à $\{3, 4, 5, 6\}$. Nous présentons maintenant les résultats numériques obtenus par les méthodes issues de l'état de l'art et avec nos algorithmes RIGA$_H$ et ILS dans la Table 2.4 ci-dessous ("/" signifie que le temps maximal d'exécution autorisé de 120 secondes est dépassé). Trois fonctions d'agrégation sont considérées : la somme pondérée, l'opérateur OWA et l'intégrale de Choquet 2-additive.

		Somme pondérée			OWA			Choquet		
méthode	p	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)
RIGA$_H$	3	11.57	14.9	0.01	5.80	4.1	0.91	15.58	23.6	0.69
	4	11.67	19.9	0.07	4.27	4.2	1.22	13.30	26.9	0.58
	5	10.45	22.2	0.10	3.75	2.7	0.46	16.67	27.9	0.80
	6	11.74	24.0	0.33	5.53	5.7	0.94	20.32	32.2	1.06
ILS	3	4.14	13.7	2.12	2.98	3.5	0.99	7.61	25.7	2.98
	4	10.70	19.8	2.01	3.01	4.8	1.17	35.67	51.4	4.06
	5	26.07	28.7	1.68	3.84	4.2	0.67	184.39	94.4	4.73
	6	28.57	34.1	1.69	4.32	3.7	1.36	772.55	152.8	4.81
NEMO	3	113.23	20.0	4.20	49.81	20.0	3.81	60.13	20.0	2.53
	4	144.35	20.0	5.05	39.96	20.0	4.39	51.02	20.0	3.07
	5	137.06	20.0	8.32	44.12	20.0	3.87	53.46	20.0	3.25
	6	149.40	20.0	9.50	53.75	20.0	4.09	58.29	20.0	2.89
IEEP$_\delta$	3	10.24	7.2	0.19	15.03	5.0	0.00	152.68	11.7	1.25
	4	24.20	12.4	0.19	23.67	5.1	0.00	/	/	/
	5	55.86	17.8	0.22	/	/	/	/	/	/
	6	173.47	23.4	0.21	/	/	/	/	/	/
Two-Phase$_\delta$	3	140.45	12.6	0.42	59.94	4.7	0.39	375.38	17.6	0.73
	4	190.61	15.6	1.28	40.19	2.4	2.63	1074.68	53.4	1.70
	5	344.10	24.8	2.13	55.90	3.0	2.51	/	/	/
	6	485.36	31.1	2.37	81.51	3.2	4.74	/	/	/

TABLE 2.4 – Résultats obtenus par RIGA$_H$, ILS, NEMO, IEEP$_\delta$ et Two-Phase$_\delta$.

Somme pondérée Pour IEEP$_\delta$ et Two-Phase$_\delta$, nous avons fixé $\delta = 0.01$ dans le but d'obtenir une valeur équivalente à l'erreur obtenue par RIGA$_H$. Tout d'abord, nous observons que RIGA$_H$ surpasse clairement ILS, NEMO et Two-Phase$_\delta$ puisqu'il est meilleur qu'eux sur tous les critères (temps, questions et erreur). Ensuite, en comparant RIGA$_H$ à IEEP$_\delta$, on constate que IEEP$_\delta$ impose moins de questions mais son erreur est plus élevée sur les plus petites instances. Sur la plus grande instance, le nombre de questions et l'erreur sont presque les mêmes, mais RIGA$_H$ est significativement plus rapide (15 fois plus rapide pour $p = 6$ critères).

6. https://polymake.org
7. https://eden.dei.uc.pt/~paquete/tsp/

OWA Ici, nous fixons $\delta = 0.02$ pour IEEP$_\delta$ et Two-Phase$_\delta$ pour les mêmes raisons que précédemment. Les erreurs sont plus importantes que celles de la somme pondérée car la recherche locale 2-opt utilisée pour optimiser un OWA avec poids connus est moins performante que l'heuristique de Lin-Kernighan utilisée pour la somme pondérée.

Ici aussi nous voyons que toutes les méthodes sauf IEEP$_\delta$ sont clairement surpassées par RIGA$_H$. IEEP$_\delta$ est assez performante sur les plus petites instances ($p = 3, 4$) en termes d'erreur et de nombre de questions, mais elle est beaucoup plus lente que RIGA$_H$ (par exemple, presque 6 fois plus lente pour $p = 4$). De plus, sur les plus grandes instances, nous observons que IEEP$_\delta$ est limité par l'utilisation d'une méthode de résolution exacte, nécessaire pour sa garantie de performance (le temps d'exécution maximal autorisé est déjà dépassé avec les critères $p = 5$).

Intégrale de Choquet 2-additive Pour IEEP$_\delta$ et Two-Phase$_\delta$, nous fixons δ à 0.02 pour les mêmes raisons que précédemment. Dans le tableau, RIGA$_H$ est clairement meilleure que toutes les autres méthodes en termes de temps de calcul, de nombre de questions et d'erreur.

Dans le tableau, nous observons que le temps maximal d'exécution permis est dépassé lors de l'exécution de IEEP$_\delta$ avec 4 critères et plus. Ceci peut s'expliquer par le fait que le nombre de points extrêmes du polyèdre représentant l'imprécision des paramètres augmente avec le nombre de critères. Par exemple, après 25 questions, le nombre de points extrêmes est approximativement égal à 4500 pour les problèmes impliquant 4 critères. Nous observons également que le temps d'exécution maximal autorisé est dépassé lors de l'application de Two-Phase$_\delta$ sur des instances à 5 critères et plus. Ceci est dû à une combinaison entre un grand nombre de solutions et au fait que le programme linéaire utilisé pour les calculs des PMR implique un nombre quadratique de variables (une par critère et une par paire de critères).

Nous pouvons donc conclure que notre algorithme RIGA est globalement le meilleur pour résoudre le TSP multi-objectifs et surpasse nettement ILS. Il peut aussi être utilisé pour résoudre des instances plus importantes, comme le montre le Tableau 2.5 avec 300 villes.

		Somme pondérée			OWA			Choquet		
	p	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)
RIGA$_H$	3	130.80	17.92	0.04	7.56	2.6	1.33	11.48	23.9	0.59
	4	171.14	20.10	0.22	5.57	4.1	1.11	15.56	26.9	0.80
	5	178.23	21.88	0.62	2.36	3.0	0.72	18.72	27.4	1.02
	6	185.27	24.06	1.12	6.19	2.6	1.37	22.78	32.4	1.34

TABLE 2.5 – Résultats obtenus par RIGA$_H$ avec 300 villes.

2.3 Application au problème de sac à dos

Dans cette section, nous montrons que RIGA obtient également de bons résultats pour un autre problème d'optimisation combinatoire multi-objectifs : le problème du sac à dos.

Le problème du sac à dos tire son nom du problème rencontré par quelqu'un qui est contraint par un sac à dos de taille fixe et qui souhaite le remplir avec ses objets les plus précieux. Le problème se pose souvent dans le cadre de l'allocation des ressources, lorsque les décideurs doivent choisir parmi un ensemble de projets ou de tâches non divisibles sous une contrainte de budget ou de temps fixe, respectivement.

2.3.1 Formalisation du problème et résolution exacte avec paramètres connus

Formellement, nous considérons un problème de décision où le décideur doit sélectionner des éléments (par exemple, des candidats, des projets, des objets) parmi un ensemble d'éléments noté $\mathcal{E} = \{1, \ldots, n\}$. Dans ce problème, tout sous-ensemble d'éléments peut être représenté par un vecteur solution $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ où $x_i = 1$ si l'élément i est dans le sous-ensemble et $x_i = 0$ sinon. Dans le problème standard du sac à dos, l'ensemble $\mathcal{X}$ des solutions admissibles est défini par des contraintes linéaires de la forme $\sum_{i=1}^{n} \tau_i x_i \leq T$ où $\tau_i \geq 0$ est le poids de l'élément i et $T \geq 0$ est le poids total maximum. Par exemple, dans les problèmes d'élection de comité, les contraintes de cardinalité sont généralement imposées pour contrôler la taille du comité élu ou pour assurer la parité hommes-femmes. Pour simplifier la présentation, nous ne considérerons qu'une seule contrainte de cardinalité, mais l'algorithme proposé peut être facilement adapté à des problèmes comportant des contraintes de faisabilité supplémentaires.

Dans le problème multi-objectifs du sac à dos, nous considérons un ensemble fini d'objectifs, à maximiser. Ainsi, à tout élément $e \in \mathcal{E}$ est associé un vecteur de performance $y(k) = (y_1(k), \ldots, y_p(k)) \in \mathbb{R}^p_+$ où $y_j(k)$ est l'évaluation de l'élément k sur le $j^{\text{ème}}$ objectif; par exemple, $y_j(k)$ peut être l'utilité du candidat k pour l'électeur j dans les problèmes d'élection de comité. Toute solution $x \in \mathcal{X}$ est alors caractérisée par un vecteur de performance $y(x) = (y_1(x), \ldots, y_j(x))$ défini par $y_j(x) = \sum_{k=1}^{n} x_k y_j(k)$ pour tous les $j \in \{1, \ldots, p\}$.

Le problème du sac à dos multi-objectifs a été utilisé dans de nombreux cas pratiques parmi lesquels on peut citer la planification des investissements dans les transports [Teng and Tzeng, 1996].

Résolution exacte avec paramètres connus

De la même manière que précédemment, dans le but d'évaluer nos méthodes, nous devons disposer d'un solver particulier pour résoudre les problèmes avec paramètres connus. Cette section est dédiée à cette tâche.

Somme pondérée Le caractère linéaire de la somme pondérée permet d'utiliser le programme linéaire classique suivant :

$$\max y_j(x) = \sum_{k=1}^{n} y_j(k)x_k \qquad \forall j \in \{1,\dots,p\}$$
$$s.t. \sum_k \tau_k x_k \leq T \qquad \forall k \in \{1,\dots,n\} \tag{2.9}$$
$$x_k \in \{0,1\} \qquad \forall k \in \{1,\dots,n\} \tag{2.10}$$

La fonction objectif maximise la somme des profits des objets choisis, le profit du $j^{\text{ème}}$ critère est remporté si l'objet k est sélectionner. La variable de décision x_k avec $k \in \{1,\dots,n\}$ vaut 1 si l'objet k est sélectionné, 0 sinon. La contrainte (2.9) limite le poids total de la sélection à T.

Si les poids des objets considérés valent 1, soient $\tau_k = 1$ pour tout $k \in \{1,\dots,n\}$, alors on peut simplement utiliser un algorithme glouton qui sélectionne itérativement les objets présentant les profits les plus élevés.

OWA Avec l'utilisation d'un OWA pour modéliser les préférences du décideur, la linéarisation est la même que celle utilisée pour le problème du voyageur de commerce, ainsi on a le programme de résolution suivant :

$$\max \sum_{i=1}^{p} (\lambda_i - \lambda_{i+1})(ir_i + \sum_{j=1}^{p} d_j^i)$$
$$s.t.\ r_i + d_j^i \geq y_j(x) \qquad \forall i,j \in \{1,\dots,p\} \tag{2.11}$$
$$d_j^i \geq 0 \qquad \forall j,i \in \{1,\dots,p\} \tag{2.12}$$
$$\sum_k \tau_k x_k \leq T \qquad \forall k \in \{1,\dots,n\} \tag{2.13}$$
$$x_k \in \{0,1\} \qquad \forall k \in \{1,\dots,n\} \tag{2.14}$$

Les contraintes (2.11) et (2.12) sont celles de la linéarisation de OWA, tandis que la contrainte (2.13) correspond au problème considérée. Si tous les poids valent 1, on peut alors remplacer la contrainte par $\sum_k x_k = T$ (au plus T objets doivent être sélectionnés).

Intégrale de Choquet 2-additive Dans ce cas, la linéarisation est également la même que celle utilisée pour le problème du voyageur de commerce.

2.3.2 Adaptation de la recherche locale et de l'algorithme génétique interactifs

Cette section présente les adaptations nécessaires à nos algorithmes pour résoudre le problème du sac à dos multi-objectifs.

ILS

Pour la solution initiale, de la même façon qu'avec le TSP, un ensemble de $m = 100$ solutions sont construites par l'utilisation de l'algorithme glouton résolvant de manière exacte le problème avec paramètres connus [Martello and Toth, 1990]. Ainsi, on détermine la solution préférée du décideur avec un seuil d'erreur $\delta_1 = 0.1$. La fonction de voisinage consiste à remplacer un élément de la solution courante par un autre élément (toutes les combinaisons sont effectuées). Nous posons ensuite des questions jusqu'à ce que le regret minimax tombe en dessous du seuil donné de $\delta_2 = 0.4$ afin de sélectionner la prochaine solution courante. Les seuils $\delta = (\delta_1, \delta_2)$ sont issus de tests exécutés de la même façon que pour le TSP, et les mêmes conclusions s'appliquent ici.

RIGA

Afin d'évaluer les performances de notre algorithme, nous testons différentes valeurs de paramètres : M le nombre d'itérations est fixé successivement à 5,10, 20, S la taille de la population est fixée à 10, 20, 30, et K le nombre de couples sélectionnés est fixé à 2, 5, 10. Pour chaque instance générée du problème du sac à dos, nous exécutons notre algorithme avec toutes les combinaisons possibles de valeurs de paramètres. Les résultats sont visibles dans le Tableau 2.6.

			Somme pondérée			OWA			Choquet		
M	S	K	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)
5	10	2	0.44	22.12	1.05	5.13	5.32	0.002	6.18	39.10	0.55
		5	0.60	25.42	0.76	4.23	5.94	0.001	4.60	42.30	0.43
		10	0.65	26.12	0.85	6.65	5.40	0.003	4.45	43.47	0.56
	20	2	0.72	29.34	0.45	7.65	6.48	0.001	6.94	50.03	0.33
		5	0.87	29.46	0.45	7.61	6.68	0.000	7.59	53.23	0.34
		10	1.17	32.28	0.46	10.25	7.36	0.001	10.95	52.15	0.32
	30	2	1.43	32.36	0.29	11.93	6.74	0.000	10.94	52.15	0.31
		5	1.56	35.38	0.38	11.99	7.64	0.000	11.21	57.08	0.34
		10	2.02	36.16	0.27	12.87	7.80	0.000	11.38	58.90	0.34
10	10	2	0.61	36.99	0.23	8.55	5.94	0.002	3.82	38.70	0.52
		5	0.71	33.58	0.32	9.98	6.20	0.001	4.59	42.30	0.43
		10	0.86	36.78	0.40	13.38	5.94	0.000	4.46	43.48	0.56
	20	2	1.05	40.50	0.10	16.03	7.16	0.001	7.34	46.72	0.35
		5	1.28	42.94	0.09	14.40	6.82	0.001	7.60	50.02	0.33
		10	1.59	42.30	0.14	19.79	7.56	0.000	17.59	53.23	0.35
	30	2	1.32	41.04	0.08	25.14	7.14	0.003	10.95	52.15	0.32
		5	1.67	45.38	0.07	23.17	8.44	0.001	11.21	57.08	0.34
		10	1.99	44.34	0.07	29.34	8.96	0.000	11.38	58.90	0.34
20	10	2	0.76	42.38	0.12	24.96	6.46	0.001	6.88	55.93	0.36
		5	0.84	42.96	0.12	25.90	6.60	0.000	6.96	57.88	0.38
		10	0.88	46.46	0.19	24.14	7.84	0.000	7.52	61.80	0.32
	20	2	1.03	46.52	0.06	31.58	7.70	0.001	17.28	61.78	0.34
		5	1.18	47.12	0.09	30.64	8.86	0.000	17.24	69.62	0.23
		10	1.62	53.02	0.08	40.41	7.32	0.000	15.32	69.18	0.30
	30	2	1.68	49.56	0.03	67.05	8.48	0.000	23.13	67.80	0.26
		5	1.93	51.84	0.03	50.42	7.68	0.000	24.49	73.01	0.29
		10	2.19	53.72	0.05	53.31	10.04	0.000	31.13	78.72	0.29

TABLE 2.6 – Résultats obtenus avec variation des paramètres M, S et K par RIGA$_H$ avec $p = 5$, $\delta = 0$, et $\mu = 0.5$.

Les valeurs $M = 10$, $S = 20$, et $K = 5$ permettent d'obtenir le meilleur compromis en termes de temps de calcul, de nombre de questions et d'erreur. En particulier, l'erreur est

trop élevée lorsque $M = 5$, alors que le nombre de questions et les temps de calcul sont prohibitifs avec $M = 20$. Pour le taux de mutation et le seuil de tolérance, nous avons réalisé des expériences similaires à celles du problème du voyageur de commerce, et nous sommes arrivés à la même conclusion : fixer μ à 0.5 et δ à 0.5 sont les meilleurs choix.

Pareillement à la section précédente, nous comparons les performances de notre algorithme face à celles obtenues par ses deux variantes : RIGA$_{KCSS}$ et RIGA$_S$ (avec un croisement en un point et une mutation par permutation). Les résultats sont donnés dans le Tableau 2.7. Logiquement, RIGA$_{KCSS}$ génère un nombre important de questions, sans réduire significativement l'erreur. Par exemple, avec $p = 5$ et une intégrale de Choquet, 134 questions sont nécessaires contre 50 pour RIGA. De plus, RIGA est beaucoup plus rapide que RIGA$_{KCSS}$. Par exemple, RIGA est environ 7 fois plus rapide que RIGA$_{KCSS}$ pour le modèle de la somme pondérée et $p = 5$ critères. Notons que l'erreur obtenue par RIGA$_S$ pourrait être réduite en augmentant le nombre M d'itérations (générations), mais cela augmenterait encore le nombre de questions (comme cela a été vu pour le problème du voyageur de commerce). En effet que RIGA$_S$ pose plus de questions que RIGA, tout en obtenant des erreurs beaucoup plus élevées.

		RIGA			RIGA$_{KCSS}$			RIGA$_S$		
fonction	p	temps (s)	# questions	erreur (%)	temps (s)	# questions	erreur (%)	temps (s)	# questions	erreur (%)
Somme pondérée	3	0.40	16.40	0.02	5.11	24.86	0.01	0.40	14.04	5.27
	5	1.28	42.94	0.09	8.40	73.72	0.04	1.57	33.20	5.39
OWA	3	5.55	5.10	0.00	12.05	7.12	0.00	0.13	7.30	3.32
	5	14.40	6.82	0.00	28.85	10.74	0.00	0.45	12.23	3.87
Choquet	3	7.03	24.14	0.64	6.36	71.26	0.64	0.47	31.60	3.93
	5	7.60	50.02	0.33	12.38	134.66	0.33	2.55	64.22	3.99

TABLE 2.7 – Résultats obtenus par RIGA, RIGA$_{KCSS}$ et RIGA$_S$ avec $\delta = 0$.

2.3.3 Résultats expérimentaux

Pareillement à la section précédente, nous comparons les performances de nos algorithmes face à celles obtenues par les méthodes existantes décrites précédemment mais adaptées au problème du sac à dos :

Necessary-preference-enhanced Evolutionary Multi-objective Optimizer (NEMO) La méthode est la même que celle décrite pour le problème du voyageur de commerce, soit basée sur un algorithme génétique qui utilise les solutions. L'opérateur de croisement correspond ici à un croisement en un point.

Méthode en deux phases (Two-Phase) Cette méthode nécessite d'obtenir l'ensemble de Pareto. Afin de le générer efficacement, nous utilisons la programmation dynamique [Bazgan et al., 2009] combinée à la méthode ND-Tree [Jaszkiewicz and Lust, 2018] pour réaliser efficacement les tests de dominance. Ensuite on sélectionne 3000 solutions aléatoirement parmi lesquelles le décideur doit déterminer sa préférée avec la stratégie CSS.

Incremental Elicitation based on Extreme Points (IEEP) Cette méthode de résolution exacte reste identique, il est simplement nécessaire d'adapter la méthode utilisée pour résoudre le problème avec paramètres connus [Benabbou and Lust, 2019a].

Résultats numériques

Dans ces expériences, nous considérons des instances du sac à dos avec 100 éléments et $p = \{3, 4, 5, 6\}$ objectifs. Tous les poids des éléments sont fixés à 1. Les instances sont générées de la façon suivante : les vecteurs de performance liés aux éléments sont uniformément tirés dans l'espace $\{1, \ldots, 1000\}^p$ et la capacité du sac à dos est fixée à 50 (la moitié du nombre d'éléments) de façon à obtenir des instances difficiles (c'est-à-dire, avec un grand nombre de solutions Pareto-optimales).

Nous comparons ici les performances de RIGA face à celles obtenues par les méthodes présentées. Les résultats sont donnés dans le Tableau 2.8. Pour IEEP$_\delta$ et Two-Phase$_\delta$, nous avons fixé $\delta = 0.01$ dans le but d'obtenir une erreur équivalente à celle obtenue par RIGA.

		Somme pondérée			OWA			Choquet		
méthode	p	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)	temps(s)	#questions	erreur(%)
RIGA	3	0.18	10.4	0.14	6.70	3.9	0.003	7.24	11.0	0.68
	4	0.24	12.0	0.21	11.18	4.3	0.001	6.39	12.1	0.31
	5	0.32	13.1	0.44	17.33	4.6	0.001	6.05	12.8	0.49
	6	0.38	14.1	0.71	28.45	4.5	0.003	7.94	14.3	0.49
ILS	3	21.83	12.6	0.05	0.32	4.0	0.08	25.23	15.6	0.89
	4	81.42	22.9	0.14	0.51	4.3	0.24	108.30	25.0	0.06
	5	273.67	34.0	0.08	2.39	12.5	0.25	516.40	38.8	0.11
	6	600.21	52.7	0.09	1.58	11.1	0.37	1056.98	49.2	0.10
NEMO	3	13.19	15.0	0.25	18.03	15.0	0.10	10.46	15.0	0.21
	4	9.45	15.0	0.93	13.99	15.0	0.22	5.35	15.0	0.56
	5	7.34	15.0	1.37	11.73	15.0	0.37	5.38	15.0	0.85
	6	5.60	15.0	1.86	10.63	15.0	0.40	5.17	15.0	1.29
IEEP$_\delta$	3	11.43	8.8	0.06	7.92	6.4	0.00	54.11	23.2	0.05
	4	16.88	13.8	0.06	9.32	7.9	0.00	/	/	/
	5	22.46	18.1	0.10	29.03	19.2	0.00	/	/	/
	6	27.46	25.1	0.19	24.99	13.5	0.05	/	/	/
Two-Phase$_\delta$	3	119.70	9.9	0.21	47.44	3.4	0.10	262.79	20.6	0.14
	4	225.02	17.4	0.12	56.54	4.2	0.26	901.03	61.9	0.10
	5	280.62	25.7	0.14	82.55	5.2	0.71	/	/	/
	6	372.18	32.7	0.19	73.90	4.5	1.15	/	/	/

TABLE 2.8 – Résultats obtenus par RIGA, ILS, NEMO, IEEP$_\delta$ et Two-Phase$_\delta$.

Somme pondérée Nous observons que RIGA est beaucoup plus rapide que toutes les autres méthodes. ILS et Two-Phase$_\delta$ sont les plus lentes. Par exemple, pour $p = 6$, RIGA est environ 1600 fois plus rapide que ILS et presque 1000 fois plus rapide que Two-Phase$_\delta$. Ceci est principalement dû au fait que le nombre de solutions non dominées augmente de façon drastique avec le nombre de critères. Bien que ces deux méthodes soient un peu plus proches

de l'optimum que RIGA, elles posent un plus grand nombre de questions que l'algorithme proposé. Par exemple, avec $p = 5$ critères, RIGA impose 14 questions contre 32 pour Two-Phase$_\delta$ et 53 pour ILS.

En comparant RIGA à IEEP$_\delta$, nous observons que ce dernier algorithme obtient des erreurs plus faibles que le premier, mais au prix de générer un nombre de questions significativement plus élevé. Il demande aussi des temps de calcul plus importants. Pour donner un exemple, pour $p = 6$, le temps de calcul est 70 fois plus faible en utilisant RIGA que IEEP$_\delta$, et le nombre de questions est divisé par environ 2.

Enfin, nous observons que RIGA est meilleur que NEMO dans tous les cas. Plus précisément, lorsque NEMO est arrêté après 15 questions (c'est-à-dire 150 générations), RIGA est beaucoup plus rapide que NEMO, produit beaucoup moins d'erreurs et demande moins de questions de préférences. Notez que le temps d'exécution de NEMO diminue lorsque le nombre de critères augmente. Cela peut s'expliquer par le fait que la construction du classement des solutions devient de plus en plus facile lorsque le nombre de critères augmente, car le nombre de solutions non dominées augmente également.

OWA Tout d'abord, nous observons que les deux méthodes basées sur les paramètres, à savoir RIGA et IEEP$_\delta$, obtiennent de très bons résultats, notamment sur la qualité de la solution retournée. Cependant, IEEP$_\delta$ doit poser beaucoup plus de questions pour fournir les garanties de performances souhaitées.

Ensuite, nous observons que ILS est l'algorithme le plus rapide, mais demande trop de questions par rapport à RIGA tout en générant des erreurs beaucoup plus importantes. Par exemple, pour $p = 5$, ILS est environ 7 fois plus rapide que RIGA, mais ILS demande environ 3 fois plus de questions et son erreur est 250 fois plus grande.

En comparant RIGA et Two-Phase$_\delta$, nous constatons que le premier surpasse le second en termes d'erreurs et de temps de calcul, tout en générant un nombre similaire de questions.

Enfin, nous observons que l'erreur obtenue par NEMO est étonnamment élevée par rapport à celle des autres méthodes, compte tenu du fait qu'elle a été autorisée à poser plus de questions de préférences.

Intégrale de Choquet 2-additive Les conclusions au sujet de la méthode IEEP$_\delta$ avec 4 critères et plus et Two-Phase$_\delta$ sur des instances à 5 critères et plus sont les mêmes. ILS est la méthode qui obtient les meilleurs résultats en termes d'erreur, mais elle pose trop de questions et est très longue par rapport à RIGA. Par exemple, avec $p = 5$ critères, RIGA soumet 13 questions, se termine après 8 secondes, et son erreur est inférieure à 0.5%, alors que ILS impose 39 questions en 516s pour obtenir une erreur légèrement inférieure (environ 0.1%).

En comparant RIGA à NEMO, on observe que le nombre de questions et le temps de calcul sont assez similaires, mais que NEMO renvoie une solution de moins bonne qualité (sauf pour $p = 3$). On peut en conclure que RIGA possède les résultats de meilleurs compromis.

2.4 Conclusion du chapitre

Nous avons proposé un algorithme de recherche locale interactif et un algortihme génétique interactif combiné à une élicitation incrémentale basée sur le regret pour résoudre des problèmes d'optimisation combinatoire multi-objectifs. Ces nouvelles méthodes, RIGA et ILS, ont été appliquées à deux problèmes combinatoires multi-objectifs : le problème du sac à dos et celui du voyageur de commerce. Selon trois indicateurs de performance (temps d'exécution, nombre de questions et erreur), RIGA surpasse plusieurs méthodes existantes et également ILS, pour différentes fonctions d'agrégation (linéaires et non linéaires). De plus, RIGA s'exécute en temps polynomial et ne demande qu'un nombre polynomial de questions.

Malgré les bons résultats expérimentaux obtenus par RIGA (erreur toujours inférieure à 1.25%), elle ne dispose d'aucune garantie théorique de performance. Dans la perspective d'obtenir un meilleur algorithme avec garantie de performance, il serait intéressant de combiner RIGA et IEEP.

Chapitre 3

Optimisation interactive d'une fonction linéaire sous contrainte de matroïde

Résumé

Nous proposons dans ce chapitre deux méthodes d'élicitation incrémentale des préférences pour l'optimisation interactive basée sur les préférences dans une structure particulière : les matroïdes pondérés (présentés en Section 3.1). Plus précisément, pour des fonctions objectifs (d'utilité) linéaires, nous proposons un algorithme interactif de type glouton qui mêle questions de préférences et construction progressive d'un ensemble indépendant pour obtenir une base optimale ou quasi-optimale d'un matroïde. Nous proposons également un algorithme interactif de recherche locale basé sur des séquences d'échanges améliorants. Ces deux algorithmes permettent de fournir des garanties de performances sur la qualité des solutions retournées et sur le nombre de questions. Nos méthodes sont ensuite testées sur les matroïdes uniforme, graphique et transversal pour résoudre trois problèmes différents : ordonnancement de tâches, élection de comité et arbre couvrant de poids minimal. Pareillement au chapitre précédent, ces algorithmes seront évalués en termes de temps de calcul, nombre de questions et pourcentage d'erreur par rapport à la solution optimale. Ce chapitre est fondé sur nos travaux présentés dans l'article intitulé "Combining preference elicitation with local search and greedy search for matroid optimization" [Benabbou et al., 2021a].

Les matroïdes ont de nombreuses applications, dans des contextes variés, et permettent de modéliser des problèmes dans plusieurs domaines tels que le recrutement, l'élection de comités, les enchères combinatoires, l'ordonnancement, l'allocation de ressources, la localisation d'installations et le placement de capteurs, pour ne citer que quelques exemples. Divers algorithmes sont maintenant disponibles pour résoudre des problèmes combinatoires sous contrainte de matroïde de manière optimale ou approximative, pour des classes spécifiques de fonctions. Dans cette thèse, nous nous intéressons au cas où la fonction à optimiser est imprécisément connue.

3.1 Optimisation dans les matroïdes

L'optimisation d'une fonction d'ensemble sous une contrainte matroïde a connu un essor depuis les années 1970 [Edmonds, 1971]. Mais les matroïdes trouvent leur origine dans un article du mathématicien H.Whitney paru en 1935 [Whitney, 1992]. Grâce à ces travaux sur le domaine de la théorie des graphes, plusieurs similitudes ont été trouvées entre les idées d'indépendance et de rang en théorie des graphes et l'indépendance linéaire et la dimension dans l'étude des espaces vectoriels. Le concept de matroïde permet de formaliser ces similitudes. Le terme "matroïde" est né de la réflexion de H.Whitney sur l'indépendance des colonnes d'une matrice. En effet, le suffixe *-oïde* sert à former des adjectifs qui signifient "qui ressemble à" indiquant alors la ressemblance avec une matrice, tout comme un astéroïde indique sa ressemble à un astre.

3.1.1 Introduction à la théorie des matroïdes

Dans cette section nous présentons quelques définitions et propositions sur les matroïdes de manière synthétique, puis nous verrons quelques exemples de matroïdes classiques. Un matroïde est composé d'un ensemble sur lequel est défini une structure d'indépendance. Plus formellement :

Definition 3.1.1 (Matroïde)**.** *Soient E un ensemble fini et $\mathcal{I}$ une collection non vide de sous-ensembles de E. Le couple $\mathcal{M} = (E, \mathcal{I})$ est un matroïde s'il vérifie les deux axiomes suivants :*

- (A_1) *si $Y \in \mathcal{I}$ et $X \subseteq Y$ alors $X \in \mathcal{I}$,*
- (A_2) *si $X \in \mathcal{I}$, $Y \in \mathcal{I}$ et $|X| < |Y|$ alors il existe $e \in Y \setminus X$ tel que $X \cup \{e\} \in \mathcal{I}$.*

Dans un matroïde $\mathcal{M} = (E, \mathcal{I})$, les éléments de $\mathcal{I}$ sont des sous-ensembles dits "indépendants". L'axiome (A_1) est appelé axiome d'hérédité. En d'autres termes, si un sous-ensemble X est contenu dans un sous-ensemble Y étant indépendant, alors X est nécessairement indépendant lui aussi. L'axiome (A_2) est l'axiome d'échange. Plus précisément, pour deux sous-ensembles X et Y indépendants avec X de taille strictement inférieure à Y, si un élément de Y non présent dans X est ajouté à X, alors le sous-ensemble obtenu est indépendant. Un sous-ensemble de E qui n'est pas indépendant, donc qui n'appartient pas à $\mathcal{I}$, est dit "dépendant".

Comme évoqué précédemment, un matroïde est une structure qui rend abstraite et généralise la notion d'indépendance linéaire dans les espaces vectoriels. On peut définir le matroïde linéaire de la façon suivante.

Soit $\mathbb{F}$ un corps et $A \in \mathbb{F}$ une matrice de dimension $k \times n$. Alors $E = \{1, 2, 3, \ldots, n\}$ contient l'ensemble des indices des colonnes de A et un ensemble indépendant de $\mathcal{I}$ contient les indices de colonnes linéairement indépendante. C'est le point de départ de la théorie des matroïdes.

Definition 3.1.2 (Circuit). *Soit $\mathcal{M} = (E, \mathcal{I})$ un matroïde. Un sous-ensemble C de E est un circuit de $\mathcal{M}$ si et seulement si :*

— $C \notin \mathcal{I}$,

— $\forall\, e \in C,\ C \,/\, \{e\} \in \mathcal{I}$.

Un circuit est un sous-ensemble dépendant minimal au sens de l'inclusion. Cela signifie que c'est un ensemble dépendant dont tous les sous-ensembles stricts sont indépendants. On note que le terme circuit trouve son origine dans la théorie des graphes. Le dual de cette notion est celle de base d'un matroïde.

Definition 3.1.3 (Base). *Soit $\mathcal{M} = (E, \mathcal{I})$ un matroïde. Un sous-ensemble B de E est une base de $\mathcal{M}$ si et seulement si :*

— $B \in \mathcal{I}$,

— $\forall\, e \in E \,/\, B,\ B \cup \{e\} \notin \mathcal{I}$.

Une base d'un matroïde correspond à un ensemble indépendant maximal au sens de l'inclusion, c'est-à-dire à un ensemble indépendant qui devient un ensemble dépendant avec l'ajout de tout autre élément de E. L'ensemble de toutes les bases d'un matroïde est noté $\mathcal{B}$. Une propriété importante et utile des bases d'un matroïde est qu'elles ont toutes la même cardinalité.

Proposition 3.1.1. *Soit $\mathcal{M} = (E, \mathcal{I})$ un matroïde. Si B_1, B_2 sont deux bases de $\mathcal{M}$, alors $|B_1| = |B_2|$.*

C'est l'axiome A_2 de la définition d'un matroïde qui implique que tous les sous-ensembles indépendants maximaux (au sens de l'inclusion) ont la même cardinalité (appelée rang de $\mathcal{M}$).

On appelle rang du matroïde $\mathcal{M}$, notée $r_{\mathcal{M}}$ ou $r(\mathcal{M})$, la cardinalité des bases d'un matroïde. Ainsi, pour tout sous-ensemble $X \subset E$ d'un matroïde $\mathcal{M} = (E, \mathcal{I})$, on peut définir la fonction de rang $r : 2^E \to \mathbb{N}$ de la façon suivante :

$$r(X) = \max_{Y \subseteq X} |Y|$$

La fonction de rang possède les propriétés suivantes [Whitney, 1992] :

Proposition 3.1.2. *Soit $\mathcal{M} = (E, \mathcal{I})$ un matroïde sur un ensemble E fini. La fonction de rang r est définie telle que :*

— $0 \leq r(X) \leq |X|, \quad \forall X \subseteq E$

— *Si Y est un sous-ensemble de E, alors $X \subseteq Y \Rightarrow r(X) \leq r(Y)$. Ce qui impose que le rang soit une fonction monotone.*
— *Pour deux sous-ensembles $X, Y \subset E$, on a $r(X \cup Y) + r(X \cap Y) \leq r(X) + r(Y)$. Ce qui signifie que la fonction de rang est sous-modulaire.*

Il existe plusieurs types de matroïdes. Dans un matroïde linéaire, point d'entrée de la théorie, la contrainte d'indépendance porte sur l'indépendance linéaire des colonnes d'une matrice, mais il existe d'autres contraintes d'indépendance. Nous présentons maintenant quelques matroïdes classiques qui seront utilisés dans les travaux de cette thèse.

Matroïde graphique

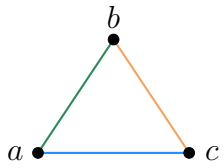
Si la première source des matroïdes est l'algèbre linéaire, la seconde est bien la théorie des graphes. En effet, les circuits des matroïdes graphiques correspondent aux cycles dans les graphes associés.

Plus précisément, on considère un graphe $G = (V, E)$ non orienté où E l'ensemble des arêtes de G qui ne contiennent aucun cycle. La collection $\mathcal{I}$ est composé des sous-ensembles d'arêtes ne contenant pas de cycle.

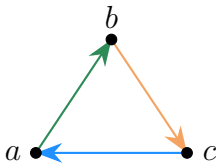
Exemple 3.1.1. *On considère dans cet exemple un matroïde graphique. Soit un graphe $G = (V, E)$ non orienté tel que les éléments de V sont les sommets et ceux de E les arêtes :*
— *$V = \{a, b, c\}$*
— *$E = \{ab, bc, ac\}$*

Ce graphe est représenté en Figure 3.1 à gauche (graphe G initial). Le matroïde $\mathcal{M} = (E, \mathcal{I})$ correspondant peut être décrit de la façon suivante :
— *E reste l'ensemble des arêtes.*
— *La collection des sous-ensembles indépendants est composé des sous-ensembles d'arêtes qui ne forment pas de cycle : $\mathcal{I} = \{\emptyset, \{ab\}, \{bc\}, \{ac\}, \{ab, bc\}, \{ab, cd\}, \{ab, ac\}, \{bc, ac\}\}$.*

Le matroïde linéaire (ou matroïde matriciel) est associé à la matrice d'incidence de G obtenue en orientant les arêtes (Figure 3.1 au centre par exemple). Une matrice d'incidence décrit un graphe par les arcs qui relient les sommets entre eux (matrice sommet-arcs dans ce cas). Dans l'exemple en Figure 3.1 on peut montrer la présence d'un cycle dans le graphe orienté G grâce à la matrice d'incidence. En effet, si on nomme les colonnes de gauche à droite c_1, c_2 et c_3 alors on a $c_3 = -c_1 - c_2$ donc les colonnes de la matrice ne sont pas indépendantes, il y a bien un cycle dans le graphe G initial.



Graphe G initial.



Graphe G orienté.

	ab	*bc*	*ac*
a	1	0	-1
b	-1	1	0
c	0	-1	1

Matrice d'incidence de G.

FIGURE 3.1 – Illustration d'un matroïde graphique.

Matroïde transversal.

Soit $E = \{e_1, \ldots, e_n\}$ et une famille $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ de sous-ensembles de E, pas nécessairement disjoints. Un transversal de $\mathcal{A}$ est un sous-ensemble T de E tel que $T = \{e_{j_1}, e_{j_2}, \ldots, e_{j_k}\}$, où $e_{j_i} \in A_i$ pour tout $i \in \{1, \ldots, k\}$.

Un ensemble $T \subseteq E$ est un transversal partiel de $\mathcal{A}$ s'il existe $\{i_1, \ldots, i_l\} \subseteq \{1, \ldots, k\}$ avec $l \leq k$ tel que T est un transversal pour la sous-famille $A_{i_1}, A_{i_2}, \ldots, A_{i_l}$ des A_i. Si $\mathcal{I}$ est défini comme l'ensemble des transversaux partiels, nous obtenons le matroïde transversal [Edmonds and Fulkerson, 1965]. Ce problème peut se représenter avec un graphe biparti.

En théorie des graphes, un graphe est dit biparti s'il existe une partition de son ensemble de sommets en deux sous-ensembles V_1 et V_2 telle qu'il y ait pour chaque arête une extrémité dans chacune des deux partitions.

Un exemple de matroïde transversal est donné dans l'exemple suivant.

Exemple 3.1.2. *Dans le graphe biparti donné en Figure 3.2(a), V_1 correspond à l'ensemble composé des sommets à gauche (correspondant à l'ensemble E, les prénoms) et V_2 de ceux à droite (les A_i).*

Une illustration simple consiste à assigner une tâche à chaque personne. Considérons l'ensemble E contenant 6 personnes $E = \{Jeanne, Serge, Olive, Tom, Jules, Vincent\}$ et $\mathcal{A}$ l'ensemble correspondant aux 4 tâches à effectuer, soit $\mathcal{A} = \{A_1, A_2, A_3, A_4\}$. Ces tâches peuvent être réalisées par les employés de l'ensemble E. Admettons que la tâche A_1 puisse être faite uniquement par Jeanne, alors on pose $A_1 = \{Jeanne\}$. On pose également $A_2 = \{Olive, Tom, Jules, Vincent\}$, $A_3 = \{Serge, Olive\}$, $A_4 = \{Serge, Tom\}$. On peut représenter cela plus simplement, sous la forme d'un graphe biparti présenté en Figure 3.2(a).

Dans ce cas on a $T = \{Jeanne, Olive, Tom, Vincent\}$ qui est un transversal de $\mathcal{A}$, présenté en Figure 3.2(b), et $X = \{Jeanne, Tom, Vincent\}$ qui est un transversal partiel de $\mathcal{A}$ car X est un transversal pour les sous-familles $\{A_1, A_2, A_4\}$.

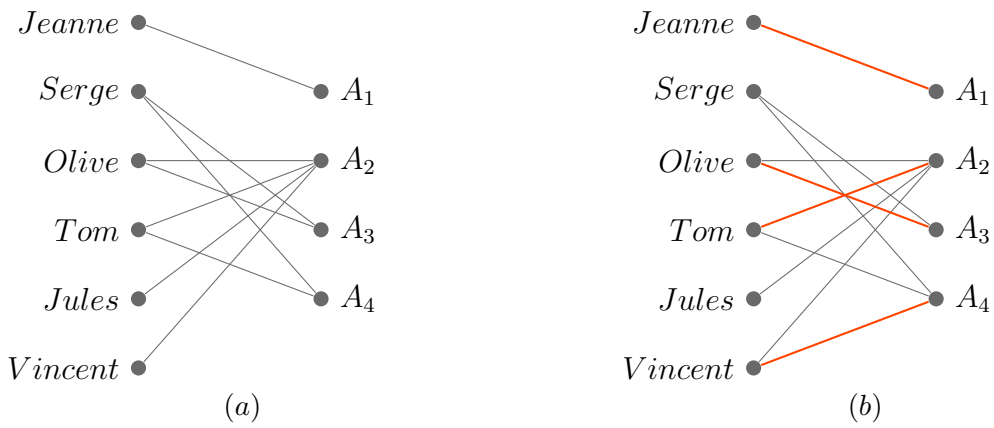


FIGURE 3.2 – Représentation dans un graphe biparti.

Ceci peut être appliqué dans une élection de comité avec différents groupes indexés par i. Dans ce cas, A_i est l'ensemble des représentants possibles du groupe i et les transversaux représentent tout ensemble de représentants appartenant à des groupes distincts.

Matroïde uniforme

Dans le cadre d'un matroïde uniforme, seuls les sous-ensembles de taille au plus k satisfont la contrainte d'indépendance, où k est un entier donné. De manière plus formelle, étant donné $E = \{1, 2, 3, \ldots, n\}$ un ensemble contenant n éléments et k un entier positif tel que $k \leq n$, la collection $\mathcal{I}$ contient les sous-ensembles de E tels que leur cardinalité est inférieure ou égale à k : $\mathcal{I} = \{X \subseteq E : |X| \leq k\}$.

Exemple 3.1.3. *On considère dans cet exemple un matroïde uniforme avec* 4 *éléments et une sélection de taille* $k = 2$. *Pour ce matroïde on a :*

- $E = \{1, 2, 3, 4\}$.
- $k = 2$.
- $\mathcal{I} = \big\{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\big\}$.
- *Tout ensemble de cardinalité* k *est une base, soit* $\mathcal{B} = \big\{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\big\}$.

Cette structure apparaît en choix social lorsque le problème est de déterminer le meilleur comité de taille k.

Matroïde de partition

Le matroïde de partition est défini à partir :

- D'un ensemble $E = \{1, \ldots, n\}$ d'éléments.
- D'une collection $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$ de m sous-ensembles disjoints de E.
- D'un entier positif $k_l \leq |\mathcal{D}_l|$ pour tout $l \in \{1, \ldots, m\}$.

Pour ce matroïde, $\mathcal{I}$ est composé de tous les sous-ensembles de E où le nombre d'éléments sélectionnés dans chaque partition $\mathcal{D}_l$ est inférieur à la valeur k_l correspondante, c'est-à-dire, $\mathcal{I} = \{X \subseteq \cup_{l=1}^{m} \mathcal{D}_l : \forall i \in \{1, \ldots, m\}, |X \cap D_l| \leq k_l\}$.

Exemple 3.1.4. *Si on considère un matroïde avec* 2 *partitions et* 5 *éléments, on peut décrire ce matroïde comme suit :*

- $E = \{e_1, e_2, e_3, e_4, e_5\}$ *et* $m = 2$,
- $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2\}$ *avec* $D_1 = \{e_3, e_4, e_5\}$ *et* $\mathcal{D}_2 = \{e_1, e_2\}$.
- *On pose* $k_1 = 2$ *et* $k_2 = 1$

La collection $\mathcal{I}$ *contient tous les sous-ensembles de* E, *contenant au maximum* 2 *éléments de* D_1 *et* 1 *de* D_2 *(donc trois au total). Plus explicitement, on a* $\mathcal{I} = \big\{\emptyset, \{e_1\}, \{e_2\}, \{e_3\}, \{e_4\}, \{e_5\}, \{e_3, e_4\}, \{e_3, e_5\}, \{e_4, e_5\}, \{e_1, e_3, e_4\}, \{e_1, e_3, e_5\}, \{e_1, e_4, e_5\}, \{e_2, e_3, e_4\}, \{e_2, e_3, e_5\}, \{e_2, e_4, e_5\}\big\}$.

L'ensemble des bases de ce matroïde, $\mathcal{B}$, *est composé de tous les ensembles indépendants maximaux, soit* $\mathcal{B} = \big\{\{e_1, e_3, e_4\}, \{e_1, e_3, e_5\}, \{e_1, e_4, e_5\}, \{e_2, e_3, e_4\}, \{e_2, e_3, e_5\}, \{e_2, e_4, e_5\}\big\}$. *Le rang du matroïde est égal à* 3.

Ces matroïdes nous permettent de modéliser de nombreux problèmes d'optimisation combinatoire comme l'arbre couvrant de poids minimum ou certains problèmes particuliers du sac à dos. L'objet de la section suivante porte sur l'optimisation sous contrainte de matroïde, dont le but est de trouver le sous-ensemble indépendant optimal pour le décideur.

3.1.2 Optimisation de matroïde pondéré

On considère le problème de la recherche d'un ensemble indépendant de poids maximum dans un matroïde. Plus précisément, étant donné un matroïde $\mathcal{M} = (E, \mathcal{I})$, nous voulons calculer :

$$\max_{X \in \mathcal{I}} w(X)$$

où w est une fonction d'ensemble définie sur 2^E donnant le poids \valeur de tout sous-ensemble de E, avec $w(\emptyset) = 0$. On suppose ici que w est monotone par rapport à l'inclusion, c'est-à-dire $\forall X \subset Y \subseteq E$ on a $w(X) \leq w(Y)$. Cette hypothèse implique que nous pouvons nous concentrer sur les bases du matroïde lors de la recherche d'un sous-ensemble indépendant optimal. Dans les problèmes de décision considérés, la fonction d'ensemble w représente les préférences subjectives du décideur. Ainsi, pour deux sous-ensembles $X, Y \in 2^E$, X est préféré à Y si et seulement si $w(X) \geq w(Y)$ (w représente l'utilité).

Dans ce chapitre, on considère le cas d'une fonction d'ensemble additive, soit caractérisée par le fait que $w(X) = \sum_{e \in X} w(\{e\})$ pour tout $X \subseteq E$, c'est-à-dire que la valeur de tout sous-ensemble est définie comme la somme des valeurs de ses éléments. La fonction w est donc entièrement caractérisée par les poids $w(\{e\})$, $e \in E$. Sans perte de généralité, nous supposons ici que $w(\{e\}) > 0$ pour tous les $e \in E$.

Dans le chapitre suivant, on s'intéressera à une fonction d'ensemble sous-modulaire. Soit un matroïde $\mathcal{M} = (E, \mathcal{I})$, la fonction d'ensemble w est dite sous-modulaire si $w(X \cup Y) + w(X \cap Y) \geq w(X) + w(Y)$ pour tout $X, Y \subseteq E$. C'est la définition classique de la sous-modularité, mais on peut aussi la définir de la façon suivante :

$$w(X \cup \{i\}) - w(X) \geq w(Y \cup \{i\}) - w(Y) \tag{3.1}$$

avec $X \subseteq Y$ et $i \notin Y$ pour tous sous-ensembles $X, Y \subseteq E$.

En d'autres termes, ajouter un élément à un sous-ensemble de taille inférieure rapporte plus que d'ajouter un élément à un sous-ensemble plus grand.

Dans les perspectives de cette thèse, on s'intéressera aux fonctions super-modulaires (une fonction w est super-modulaire si $-w$ est sous-modulaire). Ainsi w est super-modulaire si et seulement si :

$$w(X \cup \{i\}) - w(X) \leq w(Y \cup \{i\}) - w(Y) \tag{3.2}$$

avec $X \subseteq Y$ et $i \notin Y$, pour tous sous-ensembles $X, Y \subseteq E$ Ajouter un élément à un sous-ensemble de taille supérieure rapporte plus que d'ajouter un élément à sous-ensemble plus petit.

L'exemple ci-dessous illustre les propriétés de ces différentes fonctions objectif.

Exemple 3.1.5. *On considère une instance du matroïde uniforme avec 5 éléments dans l'ensemble E, et une sélection de taille $k = 3$. On s'attache à préparer des affaires pour une randonnée. On a le choix entre un imperméable rouge, un imperméable vert, une carte, une boussole et un appareil photo. L'ensemble des bases de ce matroïde, noté $\mathcal{B}$, se compose alors de 10 sous-ensembles de taille 3 (le rang du matroïde).*

Si la valeur du sous-ensemble $X = \{carte, boussole, appareil\ photo\}$ est égale à $w(X) = w(\{carte\}) + w(\{boussole\}) + w(\{appareil\ photo\})$ et ce pour tous sous-ensembles, alors la fonction d'ensemble est linéaire additive. En effet, une fonction linéaire additive impose que la valeur d'un sous-ensemble soit égale à la somme des utilités des éléments présents dans le sous-ensemble.

Soient deux sous-ensembles $X = \{appareil\ photo\}$ et $Y = \{appareil\ photo, imperméable\ vert, boussole\}$. Dans cette exemple, si l'élément $i = \{imperméable\ bleu\}$ est ajouté à l'ensemble X, le gain sera au moins égal à celui réalisé s'il est ajouté à Y. En effet, dans le sous-ensemble Y on dispose déjà d'un imperméable, peu importe la couleur la fonction est déjà remplie, ce qui n'est pas le cas pour le sous-ensemble X. Si tous les sous-ensembles respectent l'inégalité (3.1), alors la fonction objectif est dite sous-modulaire.

Soient deux sous-ensembles $X = \{appareil\ photo\}$ et $Y = \{appareil\ photo, imperméable\ vert, boussole\}$. Dans cette exemple, si l'élément $i = \{carte\}$ est ajouté à l'ensemble Y, le gain sera au moins égal à celui réalisé s'il est ajouté à X. En effet, une carte et une boussole fonctionnent en association. Par conséquent, l'ajout de la carte dans le sous-ensemble Y renforce le gain par complémentarité. Si tous les sous-ensembles respectent l'inégalité (3.2), alors la fonction objectif est dite super-modulaire.

Remarquons que l'exemple est utilisé uniquement pour présenter les trois différentes nuances car dans ce cas la fonction w n'est ni linéaire, ni sous-modulaire, ni super-modulaire (en effet, nous avons considéré parfois des égalités entre les sous-ensembles et parfois des inégalités dans les deux sens).

La section suivante est dédiée à la présentation de quelques méthodes de résolution issues de la littérature. Ces algorithmes ont été développés pour le cas classique, c'est-à-dire quand la fonction w est précisément connue.

3.1.3 Algorithmes de résolution classique

Dans cette section, on s'intéresse à la détermination d'un sous-ensemble indépendant optimal. On présente deux approches standard : approche gloutonne et approche locale.

Algorithme glouton

Lorsque la fonction d'ensemble est additive, il est bien connu que le problème peut être résolu efficacement par un algorithme glouton [Edmonds, 1971].
Cet algorithme (donné en Algorithme 5) peut se décrire de la façon suivante :

- Tant qu'un élément peut être ajouté dans l'ensemble en construction (initialement vide) en préservant l'indépendance, on itère les instructions suivantes :
  - On sélectionne un élément de E de poids maximum, parmi ceux dont l'ajout au sous-ensemble en construction conduit à un sous-ensemble indépendant.
  - L'élément sélectionné est ensuite retiré de l'ensemble E.

On peut considérer que la vérification de l'indépendance d'un sous-ensemble se fait en temps polynomial, souvent réalisée dans la littérature par un appel à un oracle.

Algorithme 5 Algorithme glouton (Edmonds, 1971)

1: $X \leftarrow \emptyset$
2: **while** E est non vide **do**
3: Sélectionner $e_i \in E$ maximisant la valeur $w(X \cup \{e_i\})$
4: **if** l'ensemble $X \cup \{e_i\} \in \mathcal{I}$ **then**
5: $X \leftarrow X \cup \{e_i\}$
6: **end if**
7: $E \leftarrow E \setminus \{e_i\}$
8: **end while**
9: **return** X

Cependant, les préférences ne sont pas toujours représentables par des fonctions additives en raison des interactions possibles entre les éléments. En théorie de la décision, l'additivité des utilités est souvent remplacée par la relâchée sous-modularité pour garantir un principe de rendements décroissants [Ahmed and Atamtürk, 2011, Lehmann et al., 2006, Vondrák, 2008] ou par la super-modularité. L'algorithme glouton présenté devient alors un algorithme d'approximation avec garantie sur la qualité des solutions retournées [Calinescu et al., 2011, Nemhauser et al., 1978, Skowron, 2017, Vondrák, 2008]. Ainsi, pour des fonctions d'ensembles monotones sous-modulaires ou super-modulaires, cet algorithme fournit une approximation de $(1 - \frac{1}{e}) \approx 0,63$ pour le matroïde uniforme et une approximation de $\frac{1}{2}$ dans le cas général [Fisher et al., 1978, Nemhauser et al., 1978].

Algorithme de recherche locale

Si on utilise une fonction d'ensemble additive, la recherche locale est un autre moyen efficace de construire des solutions optimales aux problèmes d'optimisation sous contrainte de matroïde [Lee, 2004]. Cette méthode est décrite en Algorithme 6.

Plus précisément, en partant d'une base arbitraire, on peut rechercher un élément qui peut être remplacé de manière rentable par un élément hors de la base, tout en préservant l'indépendance. En effet, il faut vérifier que chaque voisin formé à l'aide de la fonction de voisinage est bien un sous-ensemble indépendant. Ce principe d'échange peut être itéré pour améliorer progressivement la base actuelle jusqu'à atteindre un optimum local. Lorsque w est précisément connu, l'algorithme de recherche locale a une garantie de performance de 1/2, même dans le cas particulier du matroïde uniforme [Fisher et al., 1978].

Algorithme 6 Recherche locale

Soit une base du matroïde déterminée aléatoirement
amélioration ← true
while amélioration **do**
  $N_X \leftarrow \{X' \in \mathcal{B} : |X \Delta X'| = 2\}$
  $S \leftarrow N_X \cup \{X\}$
  Sélectionner $X' \in S$ maximisant la valeur $w(X')$
  if $w(X') \leq w(X)$ **then**
    amélioration ← false
  else
    $X \leftarrow X'$
  end if
end while
retourne X

La section suivante présente notre algorithme de recherche gloutonne combiné à l'élicitation incrémentale des préférences d'un décideur.

3.2 Un algorithme glouton interactif

Comme nous avons pu le voir dans la section précédente, les matroïdes présentent un intérêt particulier pour la recherche gloutonne quand w est additive : ce sont les seules structures héréditaires (vérifiant l'axiome d'hérédité noté A_1) non vides pour lesquelles l'algorithme glouton fournit des solutions optimales [Oxley, 2006].

Dans cette section, nous proposons une adaptation de l'algorithme glouton quand w n'est pas connu.

3.2.1 Description de l'algorithme

Pour pouvoir appliquer l'algorithme glouton, il faut connaître l'ordre induit par la fonction d'ensemble w (pour savoir quel élément ajouter à l'ensemble en construction à chaque itération). Quand w n'est pas connue, il s'agit de poser des questions au décideur de sorte à connaitre cet ordre (ou au moins une bonne approximation). Dans un souci d'efficacité, l'élicitation des préférences sera effectuée pendant la construction de la solution par la méthode gloutonne. L'objectif est de concentrer la tâche d'élicitation sur les informations de préférence qui sont directement nécessaires à la mise en œuvre de l'algorithme. Nous proposons une version interactive fondée sur le minimax regret de l'algorithme glouton pour l'optimisation des matroïdes (voir Algorithme 7).

Algorithme 7 Algorithme glouton interactif

1: $X \leftarrow \emptyset$;
2: **while** $|X| < r(\mathcal{M})$ **do**
3: **while** $\mathrm{MMR}(E, W) > \delta_i$ **do**
4: Demander au décideur de comparer deux éléments de E
5: Mettre à jour W selon les préférences du décideur
6: Sélectionner $e_i \in E$ tel que $\mathrm{MR}(\{e_i\}, E, W) \leq \delta_i$
7: **end while**
8: **if** $X \cup \{e_i\} \in \mathcal{I}$ **then**
9: $X \leftarrow X \cup \{e_i\}$
10: **end if**
11: $E \leftarrow E \setminus \{e_i\}$
12: **end while**
13: **return** X

Dans cet algorithme, nous utilisons les regret suivants :

Definition 3.2.1 (Minimax Regret)**.** *Pour tous ensembles $X, X' \subseteq E$ et un ensemble $\mathcal{E}$ de sous-ensembles de E :*

$$PMR(X, X', W) = \max_{w \in W} \{w(X') - w(X)\} \tag{3.3}$$

$$MR(X, \mathcal{E}, W) = \max_{X' \in \mathcal{I}} PMR(X, X', W) \tag{3.4}$$

$$MMR(\mathcal{E}, W) = \min_{X \in \mathcal{E}} MR(X, \mathcal{E}, W) \tag{3.5}$$

où W est l'ensemble des w compatibles avec les données de préférences disponibles.

Les regrets sont utilisés pour comparer des éléments deux à deux afin de pouvoir identifier le prochain élément à ajouter dans la solution X.

Dans cet algorithme, la ligne 2 utilise le rang $r(\mathcal{M})$ du matroïde pour réduire le nombre d'itérations et surtout le nombre de questions. Dans l'algorithme classique (avec w connu cf Algorithme 5), on teste tous les éléments jusqu'à ce que l'ensemble E soit vide car on pouvait itérer sans détériorer la complexité en termes de temps et de nombre de questions. Puisque nous savons que si le sous-ensemble X est de taille $r(\mathcal{M})$ alors aucun autre élément ne peut être ajouté sans que le sous-ensemble correspondant ne devienne dépendant, donc il n'est plus nécessaire de continuer à tester les éléments de E. Le rang peut être facilement obtenu pour plusieurs matroïdes comme le matroïde uniforme où k correspond aussi au rang du matroïde car il correspond au nombre maximal d'éléments qu'un sous-ensemble puisse contenir sans être dépendant. Pour des matrices plus complexes, nous pouvons utiliser n'importe quelle limite supérieure, ou précalculer $r(\mathcal{M})$ en exécutant l'algorithme glouton standard pour une fonction arbitraire w. En effet, on sait que les bases d'un matroïde ont toute la même cardinalité.

Quant à la ligne 3, le minimax regret est utilisé pour réduire le nombre de questions posées au décideur, tout en ayant une garantie sur la qualité de la solution. Ici, on a utilisé $\delta_i = \frac{\delta}{r(\mathcal{M})}$

où δ est la tolérance sur l'erreur fixée avant l'exécution de la procédure. La boucle interne (lignes 3-7) s'arrête après un nombre fini d'étapes puisque $MMR(E, W)$ est égal à 0 lorsque tous les éléments de E ont été comparés par le décideur. Notons également que l'algorithme 7 avec $\delta = 0$ sélectionne un élément qui est nécessairement optimal à chaque étape, c'est-à-dire que l'élément sélectionné est optimal pour toutes les fonctions d'ensemble admissibles $w \in W$. Par conséquent, notre algorithme avec $\delta = 0$ n'est rien d'autre que l'algorithme glouton standard [Edmonds, 1971], et il retourne une base optimale. Cependant, poser des questions jusqu'à ce que les regrets minimax diminuent à zéro peut entraîner un nombre important de questions en pratique, comme on le verra dans la Section 3.4. Pour une mise en œuvre de cette procédure, l'Algorithme 7 est utilisé avec $\delta > 0$ en pratique pour construire une base avec un regret borné.

Plus précisément, dans la section suivante, nous montrons que le regret de la solution retournée par notre algorithme glouton est bornée par la valeur δ.

3.2.2 Garanties de performances : qualité de la solution retournée, temps et nombre de questions

Proposition 3.2.1. *L'algorithme 7 renvoie une base $B \in \mathcal{B}$ telle que $MR(B, \mathcal{B}, W) \leq \delta$ soit vrai à la fin de l'exécution.*

Démonstration. Nous voulons prouver qu'à la fin de la procédure, on a bien $MR(B, \mathcal{B}, W) \leq \delta$. Par définition des max regrets, il suffit de prouver que $w(B^*) - w(B) \leq \delta$ est vrai pour toute base $B^* \in \mathcal{B}$ et pour toute fonction w qui est encore admissible à la fin de l'exécution. Pour cela, prouvons par induction que l'affirmation suivante, notée $P(i)$, est vérifiée à la fin de l'itération $i \in \{1, \ldots, r(m)\}$ de la boucle externe (lignes 2-12) :

$$\exists B_i \in \mathcal{B} \text{ tel que } \begin{cases} X_i \subseteq B_i & \text{(3.6a)} \\ B_i \backslash X_i \subseteq E_i & \text{(3.6b)} \\ w(B^*) - w(B_i) \leq |X_i| \times \dfrac{\delta}{r(\mathcal{M})} & \text{(3.6c)} \end{cases}$$

où X_i (resp. E_i) désigne l'ensemble X (resp. E) à la fin de l'itération i. En d'autres termes, nous voulons prouver que X_i peut être étendu en une base B_i possédant un regret inférieur à $|X_i| \times \frac{\delta}{r(\mathcal{M})}$. Pour l'itération $i = 0$ (avant d'entrer dans la première boucle), nous avons $X_i = \emptyset$ et $E_i = E$. Par conséquent, si nous fixons $B_i = B^*$, alors les équations (3.6a-3.6c) sont évidemment satisfaites. Par conséquent, $P(0)$ est vérifiée.

Maintenant, on suppose que $P(i-1)$ est valable pour une certaine étape i dans $\{1, \ldots, r(m)\}$ et nous voulons prouver que $P(i)$ est vraie. En d'autres termes, en supposant que les équations (3.6a-3.6c) soient vraies pour une base B_{i-1}, nous devons identifier une base B_i telle que les équations (3.6a-3.6c) soient satisfaites par B_i. D'après la ligne 8, deux cas peuvent se produire : soit $X_{i-1} \cup \{e_i\} \notin \mathcal{I}$ soit $X_{i-1} \cup \{e_i\} \in \mathcal{I}$.

Cas $X_{i-1} \cup \{e_i\} \notin \mathcal{I}$: Dans ce cas, nous avons $X_i = X_{i-1}$ (voir lignes 8-9). Prouvons qu'il suffit de poser $B_i = B_{i-1}$ pour établir le résultat. Tout d'abord, notons que l'hypothèse

d'induction (HI) implique directement que les équations (3.6a) et (3.6c) sont vraies pour B_i puisque $X_i = X_{i-1}$ et $B_i = B_{i-1}$. Ensuite, pour l'équation (3.6b), nous devons prouver que nous avons $B_i \backslash X_i \subseteq E_i$, c'est-à-dire $B_{i-1} \backslash X_{i-1} \subseteq E_i$. Par l'HI, on a $B_{i-1} \backslash X_{i-1} \subseteq E_{i-1}$, c'est-à-dire $B_{i-1} \backslash X_{i-1} \subseteq E_i \cup \{e_i\}$ (voir ligne 9). De plus, nous pouvons déduire que $e_i \notin B_{i-1}$ de l'axiome A_1 puisque $X_{i-1} \cup \{e_i\} \notin \mathcal{I}$ et $X_{i-1} \subseteq B_{i-1}$ (par l'HI). Donc $B_{i-1} \backslash X_{i-1} \subseteq E_i$ est vérifié et donc $P(i)$ est vraie.

Cas $X_{i-1} \cup \{e_i\} \in \mathcal{I}$: On a ici $X_i = X_{i-1} \cup \{e_i\}$ (voir lignes 8-9) et on peut alors distinguer deux cas :

— Cas $e_i \in B_{i-1}$: Montrons que l'on peut simplement fixer $B_i = B_{i-1}$.
Plus précisément, pour l'équation (3.6a), nous avons :

$$\begin{aligned} X_i &= X_{i-1} \cup \{e_i\} && \text{(par hypothèse)} \\ &\subseteq B_{i-1} \cup \{e_i\} && \text{(par l'HI)} \\ &= B_{i-1} && \text{(puisque } e_i \in B_{i-1}\text{)} \\ &= B_i && \text{(par définition)} \end{aligned}$$

Pour l'équation (3.6b), nous avons :

$$\begin{aligned} B_i \backslash X_i &= B_{i-1} \backslash (X_{i-1} \cup \{e_i\}) \\ &\subseteq E_{i-1} \backslash \{e_i\} && \text{(par l'HI)} \\ &= E_i && \text{(voir ligne 9)} \end{aligned}$$

Pour l'équation (3.6c), nous avons :

$$\begin{aligned} w(B^*) - w(B_i) &= w(B^*) - w(B_{i-1}) && \text{(par définition)} \\ &\leq |X_{i-1}| \times \frac{\delta}{r(\mathcal{M})} && \text{(par l'HI)} \\ &\leq |X_i| \times \frac{\delta}{r(\mathcal{M})} && \text{(puisque } |X_i| = |X_{i-1}| + 1\text{)} \end{aligned}$$

Donc $P(i)$ est vrai.

— Cas $e_i \notin B_{i-1}$: Notons que $|X_i| \leq |B_{i-1}|$ puisque B_{i-1} est une base. Donc en appliquant itérativement l'axiome A_2, on peut conclure qu'il existe $Y \subseteq B_{i-1} \backslash X_i$ tel que $|X_i \cup Y| = |B_{i-1}|$ et $X_i \cup Y \in \mathcal{I}$. Maintenant, nous fixons $B_i = X_i \cup Y$ et nous voulons prouver que les équations (3.6a-3.6c) sont satisfaites par B_i. Notons que l'équation (3.6a) est évidemment vraie puisque $B_i = X_i \cup Y$ (par définition). Pour l'équation (3.6b), puisque $B_i \backslash X_i = Y$, nous avons seulement besoin de prouver que $Y \subseteq E_i$.
On a :

$$\begin{aligned} Y &\subseteq B_{i-1} \backslash X_i && \text{(par définition)} \\ &= B_{i-1} \backslash (X_{i-1} \cup \{e_i\}) && \text{(puisque } X_i = X_{i-1} \cup \{e_i\}\text{)} \\ &\subseteq E_{i-1} \backslash \{e_i\} && \text{(par l'HI)} \\ &= E_i && \text{(voir ligne 9)} \end{aligned}$$

Pour l'équation (3.6c), nous devons prouver que l'inégalité $w(B^*) - w(B_i) \leq |X_i| \times \frac{\delta}{r(\mathcal{M})}$ est vérifiée. On a :

$$\begin{aligned}
B_i \backslash B_{i-1} &= (X_i \cup Y) \backslash B_{i-1} && \text{(par définition)} \\
&= (X_{i-1} \cup \{e_i\} \cup Y) \backslash B_{i-1} && \text{(par hypothèse)} \\
&= (\{e_i\} \cup Y) \backslash B_{i-1} && \text{(puisque } X_{i-1} \subseteq B_{i-1} \text{ par l'HI)} \\
&= \{e_i\} && \text{(puisque } Y \subseteq B_{i-1} \text{ et } e_i \notin B_{i-1})
\end{aligned}$$

En conséquence, puisque nous avons également $|B_i| = |B_{i-1}|$, nous savons qu'il existe $e_j \in E \backslash \{e_i\}$ tel que $B_i = (B_{i-1} \cup \{e_i\}) \backslash \{e_j\}$.
Prouvons maintenant que $e_j \in E_{i-1}$:

$$\begin{aligned}
\{e_j\} &= B_{i-1} \backslash B_i = B_{i-1} \backslash (X_i \cup Y) \\
&= B_{i-1} \backslash (X_{i-1} \cup \{e_i\} \cup Y) \subseteq B_{i-1} \backslash X_{i-1} \subseteq E_{i-1} && \text{(par l'HI)}
\end{aligned}$$

Nous pouvons donc obtenir que $w(e_i) - w(e_j) \leq \frac{\delta}{r(\mathcal{M})}$ à l'aide de la définition des max regrets (voir ligne 7). Par conséquent, nous avons :

$$\begin{aligned}
w(B^*) - w(B_i) &= w(B^*) - w((B_{i-1} \cup \{e_i\}) \backslash \{e_j\}) && \text{(par définition)} \\
&= w(B^*) - w(B_{i-1}) + (w(\{e_i\}) - w(\{e_j\})) && \text{(par additivité)} \\
&\leq |X_{i-1}| \times \frac{\delta}{r(\mathcal{M})} + (w(\{e_i\}) - w(\{e_j\})) && \text{(par l'HI)} \\
&\leq (|X_{i-1}| + 1) \times \frac{\delta}{r(\mathcal{M})} && (\text{car } w(e_i) - w(e_j) \leq \tfrac{\delta}{r(\mathcal{M})}\ ) \\
&= |X_i| \times \frac{\delta}{r(\mathcal{M})} && (\text{car } X_i = X_{i-1} \cup \{e_i\})
\end{aligned}$$

Donc l'équation (3.6c) est vraie et par conséquent $P(i)$ est vraie.

Ainsi, l'affirmation $P(i)$ est vraie pour chaque étape $i \in \{1, \ldots, m\}$, et en particulier $P(i)$ est vraie pour i. Puisque X est une base, alors nous avons nécessairement $X = B$ en raison de l'équation (3.6a). Ensuite, en utilisant l'équation (3.6c), on obtient $w(B^*) - w(X) \leq |X| \times \delta / r(\mathcal{M}) = r(\mathcal{M}) \times \frac{\delta}{r(\mathcal{M})} = \delta$. Enfin, puisque l'Algorithme 7 renvoie $B = X$, le résultat est établi.

□

Il est important de noter également que l'algorithme 7 ne génère pas plus qu'un nombre polynomial de questions puisque dans le pire des cas, on demande au décideur de comparer tous les éléments de E (voir ligne 4). Par conséquent, le nombre total d'étapes de la boucle interne (lignes 3-7) est également polynomial. Cependant, malgré l'existence d'un formalisme général, l'implémentation de l'algorithme 7 peut différer de manière significative d'un contexte d'application à l'autre.

Plus précisément, vérifier l'indépendance d'un ensemble (si $X \cup \{e_i\} \in \mathcal{I}$) peut être plus ou moins complexe selon le matroïde considéré. Les trois matroïdes utilisés dans la Section 3.4 (de tests numériques) soient les matroïdes uniforme, graphique et transversal, possèdent un test d'indépendance qui peut être effectué en temps polynomial.

De plus, le calcul des regrets peut être plus ou moins complexe selon les hypothèses faites sur w. À cet égard, une option intéressante consiste à définir w par une fonction paramétrique qui est linéaire en ses paramètres. Cela peut être une combinaison linéaire de fonctions splines, une utilité linéaire multi-attributs ou une moyenne pondérée ordonnée de valeurs de critères. Dans ce cas, l'optimisation des regrets peut être effectuée en temps polynomial en utilisant la programmation linéaire. De plus, l'utilisation d'une définition paramétrique de w permet d'économiser de questions car toute déclaration de préférence du type $w(e) > w(e')$ se traduit par une contrainte sur l'espace des paramètres qui réduira les préférences possibles sur d'autres éléments.

Comme on vient de le voir, les matroïdes présentent bien un intérêt particulier pour la recherche gloutonne, même quand les préférences sont imprécisément connues. Mais la recherche locale tire aussi profit des structures de type matroïde.

3.3 Un algorithme de recherche locale interactive

Comme nous l'avons déjà vu, la recherche locale est une approche heuristique très générale utilisée de manière standard en recherche opérationnelle pour déterminer des solutions de bonne qualité à des problèmes d'optimisation combinatoire difficiles [Russell and Norvig, 2002, E.H. Aarts, 2003]. Plusieurs algorithmes de recherche locale sont décrits dans la Section 1.1.4. On s'intéresse ici à la recherche locale de type hill climbing, qui consiste à choisir la solution voisine qui optimise localement la fonction w dans le voisinage.

Dans de nombreux cas, la recherche locale fournit des solutions sous-optimales au problème global et la méthode de recherche doit être modifiée pour poursuivre la recherche au-delà de l'optimalité locale. Cependant, il existe des exemples bien connus de problèmes pour lesquels les algorithmes de recherche locale fournissent des solutions optimales. Par exemple, en optimisation continue, l'algorithme du simplexe utilisé pour résoudre des programmes linéaires est essentiellement un algorithme de recherche locale passant des sommets d'un polytope aux sommets voisins jusqu'à atteindre un optimum local, qui est connu pour être un optimum global. Dans le domaine de l'optimisation discrète, le problème de l'arbre couvrant minimum peut également être résolu par recherche locale à partir de n'importe quel arbre couvrant initial arbitraire en utilisant des voisinages et des séquences d'amélioration basées sur des échanges d'arêtes. Ce résultat est fortement lié à la justesse de l'algorithme glouton dans le problème de l'arbre couvrant (algorithme de Kruskal [Kruskal, 1956]) et plus généralement à la structure d'un matroïde pondéré.

Dans un problème d'optimisation combinatoire modélisé à l'aide d'une structure de matroïde pondéré où les solutions admissibles sont des bases, l'Algorithme 6 de recherche locale peut en effet être utilisé. Dans ce cas, on peut avoir comme point de départ n'importe quelle base que l'on améliore progressivement en utilisant des échanges d'éléments jusqu'à atteindre la base optimale, comme suggéré dans l'article [Lee, 2004] avec un algorithme d'échange simple. Pour effectuer des échanges utiles dans une recherche locale, nous devons connaître les valeurs relatives des éléments composant les solutions réalisables.

Lorsque cette information sur les préférences n'est pas disponible au début de la recherche,

elle peut être acquise au cours de la recherche via des procédures d'élicitation des préférences afin de réduire la charge d'élicitation. C'est ce que nous proposons dans la section suivante.

3.3.1 Description de l'algorithme

Lorsque w n'est pas connu, le principe de recherche locale peut être combiné avec un algorithme d'élicitation des préférences pour collecter les informations nécessaires à l'identification des échanges améliorants. Pour mettre en œuvre cette idée, nous proposons l'Algorithme 8 où $X \Delta Y$ désigne la différence symétrique définie par $(X \setminus Y) \cup (Y \setminus X)$ et N_B désigne le voisinage de la base B, c'est-à-dire l'ensemble des bases adjacentes qui diffèrent de B par exactement deux éléments (on retire un élément pour en mettre un autre).

Algorithme 8 Algorithme de recherche locale interactive

1: $B \leftarrow$ `GenerationBaseInitiale` (M)
2: amélioration $\leftarrow$ **true**
3: **while** amélioration **do**
4: $N_B \leftarrow \{B' \in \mathcal{B} : |B \Delta B'| = 2\}$
5: **while** $\text{MMR}(N_B \cup \{B\}, W) > \frac{\delta}{r(\mathcal{M})}$ **do**
6: Demander au décideur de comparer deux bases dans $N_B \cup \{B\}$
7: Mettre à jour W selon les préférences du décideur
8: **end while**
9: **if** $\text{MR}(B, N_B, W) \leq \frac{\delta}{r(\mathcal{M})}$ **then**
10: amélioration $\leftarrow$ false
11: **else**
12: $B \leftarrow$ `Select`$(\arg \min_{B' \in N_B} MR(B', N_B \cup \{B\}, W))$
13: **end if**
14: **end while**
15: **return** B

La procédure `GenerationBaseInitiale` , appelée à la ligne 1, peut être n'importe quelle heuristique fournissant une bonne solution de départ. Par exemple, on peut calculer une base optimale pour un échantillon de fonctions d'ensemble arbitraires et demander au décideur de nous dire laquelle est la meilleure. On effectue ensuite le voisinage de cette solution, la solution courante, puis dans la boucle interne (lignes 5-8), on détermine la base préférée du décideur dans l'ensemble composé de la base courante et de son voisinage. À l'aide de ces nouvelles connaissances sur les préférences du décideur, on vérifie en ligne 9 si le regret maximal de cette solution face à tout son voisinage est inférieur à une certaine valeur, soit $\delta_i = \frac{\delta}{r(\mathcal{M})}$. Si tel est le cas, on place la valeur booléenne `improve` à false, qui indique que l'étape n'a pas été améliorante et qu'on peut sortir de la boucle principale (lignes 3-12). Puis, on retourne cette base. Sinon, on réitère les mêmes opérations avec la nouvelle solution courante qui est une base du voisinage minimisant le max regret.

Tout comme l'algorithme présenté dans la section précédente, la recherche locale possède

des garanties de performances sur la qualité de la solution retournée, détaillées dans la sous-section suivante.

3.3.2 Garanties de performances : qualité de la solution retournée

La qualité de la base retournée par l'algorithme 8 est garantie par ce qui suit :

Proposition 3.3.1. *Si $B \in \mathcal{B}$ est une base retournée par l'algorithme 8, alors $MR(B, \mathcal{B}, W) \leq \delta$ est vrai.*

Démonstration. Nous voulons prouver que $MR(B, \mathcal{B}, W) \leq \delta$ est vrai lorsque l'algorithme s'arrête. Par définition des max regrets, nous avons seulement besoin de prouver que $w(B^*) - w(B) \leq \delta$ est vrai pour toute base $B^* \in \mathcal{B}$ et pour toute fonction $w \in W$ qui est encore admissible à la fin de l'exécution.

En raison du théorème des échanges multiples [Greene and Magnanti, 1975], on sait qu'il existe une bijection définie par $\sigma : B \to B^*$ telle que $B_i = (B \backslash \{e_i\}) \cup \{\sigma(e_i)\}$ est une base du matroïde pour chaque élément $e_i \in B$. Notons que $B_i \in N_B$ pour tout $e_i \in B$ puisque nous avons $|B \Delta B_i| = 2$ (voir ligne 4). Notons également qu'on a nécessairement $MR(B, N_B, W) \leq \frac{\delta}{r(\mathcal{M})}$ à la fin de l'exécution (ligne 9). Par conséquent, $w(B_i) - w(B) \leq \frac{\delta}{r(\mathcal{M})}$ est vrai par définition du max regret. Alors, puisque w est additive et que $B_i = (B \backslash \{e_i\}) \cup \{\sigma(e_i)\}$, on obtient finalement $w(\sigma(e_i)) - w(e_i) \leq \frac{\delta}{r(\mathcal{M})}$.

Ainsi, on obtient :

$$\begin{aligned} w(B^*) - w(B) &= w\Big(\bigcup_{i=1}^{r(\mathcal{M})} \{\sigma(e_i)\}\Big) - w(B) && \text{(par définition de } \sigma) \\ &= \sum_{i=1}^{r(\mathcal{M})} \Big(w(\sigma(e_i)) - w(e_i)\Big) && \text{(par additivité de } w) \\ &\leq \sum_{i=1}^{r(\mathcal{M})} \frac{\delta}{r(\mathcal{M})} = \delta \end{aligned}$$

De ce fait, on a bien $MR(B, \mathcal{B}, W) \leq \delta$ à la fin de l'exécution.

□

Lorsque $\delta = 0$, la proposition 3.3.1 assure que l'algorithme 8 produit une base optimale. Notons également qu'il ne génère pas plus qu'un nombre polynomial de questions lorsque w est additif, car calculer les valeurs PMR entre deux voisins revient à comparer deux éléments de l'ensemble E.

Par contre, contrairement à l'algorithme glouton, nous ne pouvons pas prouver que l'algorithme de recherche locale génère un nombre polynomial de questions et se termine après un nombre polynomial d'itérations. Certains cycles peuvent en effet se produire comme illustrer dans l'Exemple 3.3.1. Cependant, lorsqu'un cycle est détecté, il peut être facilement brisé en divisant itérativement δ par deux (tout en garantissant la quasi-optimalité de la base retournée).

Exemple 3.3.1. *Imaginons une instance d'un problème où* $\mathcal{B} = \{B_1, B_2, B_3, B_4\}$. *Soient* $y(B_1) = (0, 4)$, $y(B_2) = (1, 3)$, $y(B_3) = (2, 2)$ *et* $y(B_4) = (3, 1)$. *On utilise une somme pondérée et on considère que l'ensemble des préférences collectées* Θ *est initialement est vide, ainsi* Λ_Θ, *correspondant aux jeux de poids admissibles* λ, *est défini par* $\Lambda_\Theta = \{\lambda \in [0, 1]^3 : \sum_{i=1}^{3} \lambda_i = 1\}$. *On fixe le seuil de tolérance* δ *à 1 et soit* $B = B_2$ *la base initiale.*

Étape 1 : *Considérons que le voisinage de* B *contient* B_3 *et* B_4, *soit* $N_B = \{B_3, B_4\}$. *Ainsi, on a* $MMR(N_B \cup B, \Lambda_\Theta) = 1$ *et la solution de minimax regret est* B_3. *Alors* $B = B_3$.

Étape 2 : *Considérons que le voisinage de* B *contient* B_1 *et* B_2, *soit* $N_B = \{B_1, B_2\}$. *Ainsi, on a* $MMR(N_B \cup B, \Lambda_\Theta) = 1$ *et la solution de minimax regret est* B_2. *Alors* $B = B_2$.

On constate que l'algorithme boucle. Cet exemple ne correspond pas à un problème réel, nous avons essayé de générer une petite instance d'un problème combinatoire sous différents types de contrainte de matroïde pour laquelle cette boucle pouvait se produire, mais aucune instance n'a pu être trouvée.

Toutefois, nous pourrons observer dans la section expérimentale les bons résultats de l'algorithme de recherche locale malgré cela. Après avoir défini les garanties théoriques de nos algorithmes, la section suivante présente les résultats numériques obtenus.

3.4 Applications

Nous considérons ici des problèmes d'optimisation multi-objectifs dans lesquels chaque élément du matroïde $e \in E$ est évalué par un vecteur $x_e \in \mathbb{R}^p$ où p est le nombre d'objectifs à maximiser. Les valeurs des critères sont tirées aléatoirement entre 1 et 1000. Nous considérons des instances avec 50 éléments et le nombre de critères p varie dans $\{4, 6, 8\}$. Les préférences du décideur sur les éléments sont définies à l'aide d'une fonction scalarisante $f_\lambda : \mathbb{R}^p \to \mathbb{R}$ (linéaire en son paramètre λ) par $w(\{e\}) = f_\lambda(e)$. Par conséquent, toute information de préférence du type "e est au moins aussi bon que e'" se traduit par la contrainte $f_\lambda(e) \geq f_\lambda(e')$ qui est linéaire en λ. Des exemples de fonctions admissibles pour f_λ sont les sommes pondérées, les moyennes pondérées ordonnées ou les intégrales de Choquet. Nous supposons ici que f_λ est une somme pondérée et que λ est initialement inconnu.

Dans nos algorithmes, nous posons des questions de préférences au décideur jusqu'à ce que les valeurs du MMR tombent en dessous de $\frac{\delta}{r(\mathcal{M})}$. Dans nos tests, deux valeurs de δ ont été utilisées : $\delta = 0$, qui garantit que la solution retournée est optimale, et $\delta = 20\%$ du regret initial, pour réduire le nombre de questions générées. Plus précisément, après que l'instance soit établie, les solutions associées aux points extrêmes du polyèdre de départ sont déterminées à l'aide d'un algorithme exact avec jeu de poids connu. Puis, on calcule le minimax regret sur ces solutions, et cela nous retourne alors le regret initial de l'instance, dont on prend 20% comme seuil pour nos algorithmes dans les tests présentés.

On précise également qu'une élimination des points non dominés est faite avant la présentation des alternatives pour le calcul du minimax regret. Pour cela on réalise des tests de Pareto dominance entre tous les points associés aux éléments de l'ensemble E_i pour le

recherche gloutonne et entre tous les points associés aux solutions de l'ensemble $N_B \cup \{B\}$ pour la recherche locale.

Pour générer des questions informatives, nous utilisons la Current Solution Strategy, (CSS) décrite en Section 1.3.1, afin de réduire efficacement l'ensemble des paramètres admissibles. Les réponses aux questions sont simulées en utilisant une fonction w^* générée à partir d'un vecteur de pondération λ^* produit aléatoirement avant chaque exécution des algorithmes.

Calcul du regret (erreur) réel Puisque l'algorithme glouton classique (présenté en Section 3.1.3) est un algorithme de résolution exacte dans le cadre d'une fonction w additive, alors il nous permet de générer la solution optimale du décideur. En effet on peut utiliser la méthode classique avec w^* pour avoir sa solution préférée. Cela permet de déterminer la distance à l'optimale de la solution retournée par nos méthodes.

Nous présentons maintenant les problèmes et les résultats obtenus.

3.4.1 Le matroïde transversal

Le matroïde transversal peut être utilisé pour modéliser différents problèmes comme l'affectation, le couplage ou encore une élection de comité. Dans cette section, on utilise le matroïde transversal pour modéliser le problème d'ordonnancement présenté dans le paragraphe suivant et avec lequel nous présenterons un exemple d'exécution de chacun des deux algorithmes interactifs que nous avons proposés en début de section.

Problème d'ordonnancement. L'ensemble E se compose de n tâches, qui prennent chacune une unité de temps de traitement. Toutes les tâches sont disponibles au temps 0 et ont une date de fin limite notée d_j. De plus, l'exécution de chaque tâche j produit un bénéfice w_j si la tâche est terminée avant le temps d_j.

Le problème consiste alors à trouver un ordre des tâches qui maximise le bénéfice total. Formellement, si $E = \{1, \ldots, n\}$ et $\mathcal{I} = \{X \subseteq E : X$ peut être exéctué à temps$\}$ alors $\mathcal{M} = (E, \mathcal{I})$ est un matroïde. Si l'on définit $w(X) = \sum_{j \in X} w_j$ pour tout $X \in \mathcal{I}$ alors toute base optimale est un ensemble de tâches maximisant $w(X)$ (le profit total).

Le test d'indépendance consiste à vérifier pour un ensemble $X \subseteq E$, s'il existe une manière d'ordonnancer les tâches de sorte qu'elles se terminent à temps. L'algorithme utilisé pour réaliser ce test est l'ordonnancement des tâches "au plus tard". C'est un algorithme itératif qui planifie la tâche courante sur sa date de fin ou, si une tâche est déjà positionnée à cette date, au premier temps disponible avant. Cet algorithme nous permet de vérifier qu'un ordonnancement est bien réalisable efficacement. Il est utilisé :

— Dans l'algorithme glouton, lors de l'ajout d'une tâche à la solution courante.
— Dans la recherche locale, lors de la conception du voisinage, dans le but de savoir si une solution voisine correspond bien à un sous-ensemble indépendant.

On illustre ce principe simple dans l'exemple suivant.

Exemple 3.4.1. *On dispose de 8 tâches $\{j_1, \ldots, j_8\}$ ayant une durée d'une unité de temps avec les dates de fin suivantes : $d_1 = 4$, $d_2 = 1$, $d_3 = 2$, $d_4 = 4$, $d_5 = 1$, $d_6 = 3$, $d_7 = 4$ et $d_8 = 1$. On considère le sous-ensemble composé de $\{j_1, j_2, j_3, j_4\}$, et on cherche à savoir s'il existe un ordonnancement admissible de ces 4 tâches (test d'indépendance).*

Dans un premier temps, on place j_1 en position 4 car cela correspond à sa date de fin, ligne (a) Figure 3.3. Ensuite de la même façon j_2 est assignée à la position 1, en ligne (b). Puis j_3 en position 2, ligne (c). Pour j_4, comme $d_4 = 4$ et que la tâche j_1 déjà en position 4, l'algorithme n'est pas en mesure de la placer sur sa date au plus tard. Dans ce cas, on recherche la première place disponible, par ordre décroissant, en partant de la date de fin de j_4. On arrive au temps 3 ici, en Figure 3.3 ligne (d).

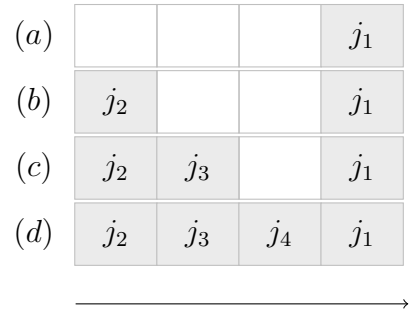


FIGURE 3.3 – Ordonnancement au plus tard.

Pour un ensemble de tâches de taille n, cet algorithme a une complexité en $\mathcal{O}(n^2)$ (n tâches à ordonnancer et n tests pour rechercher la position de la tâche au maximum).

L'exemple suivant est une exécution de notre algorithme glouton interactif présenté en Section 3.2.1 sur le problème d'ordonnancement modélisée par le matroïde transversal.

Exemple 3.4.2. *Considérons une instance du problème d'ordonnancement avec $E = \{1, \ldots, 8\}$ (8 tâches de longueur unitaire). L'utilité $w(j)$ de terminer la tâche j à temps est définie à partir d'un vecteur performance à trois critères $y(j) = (y_1(j), y_2(j), y_3(j))$ comme suit*

$$w(j) = \lambda_1 y_1(j) + \lambda_2 y_2(j) + \lambda_3 y_3(j) \tag{3.7}$$

pour un certain paramètre inconnu $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ représentant le système de valeurs du décideur. Les vecteurs de performances des tâches sont donnés dans le Tableau 3.1, ainsi que leurs échéances d_j :

	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8
$y_1(j)$	6	2	5	8	1	6	3	2
$y_2(j)$	8	4	2	7	2	3	4	3
$y_3(j)$	8	7	5	1	8	3	6	1
d_j	4	1	2	4	1	3	4	1

TABLE 3.1 – Vecteurs de performances et dates de fin des tâches.

Initialement l'ensemble Λ_Θ correspondant aux jeux de poids admissibles λ (selon l'ensemble des préférences collectées Θ) est défini par $\Lambda_\Theta = \{\lambda \in [0,1]^3 : \sum_{i=1}^{3} \lambda_i = 1\}$ et induit un ensemble W de fonctions admissibles w par l'équation (3.7). Pour simuler les réponses du décideur pendant l'exécution de l'Algorithme 7 (glouton interactif), nous supposons que la fonction d'ensemble w du décideur est définie avec $\lambda^* = (\frac{6}{9}, \frac{2}{9}, \frac{1}{9})$. Dans la Figure 3.4, Λ_Θ est représenté par le triangle ABC dans l'espace (λ_1, λ_2), λ_3 étant implicitement défini par $\lambda_3 = 1 - \lambda_1 - \lambda_2$, et λ^* est représenté par un point à l'intérieur du triangle. Dans un premier temps, nous avons appliqué l'algorithme glouton standard (défini en Section 3.1.3) sur cette même instance, avec un jeu de poids λ arbitraire et nous avons obtenu le rang du matroïde, soit $r(\mathcal{M}) = 4$.

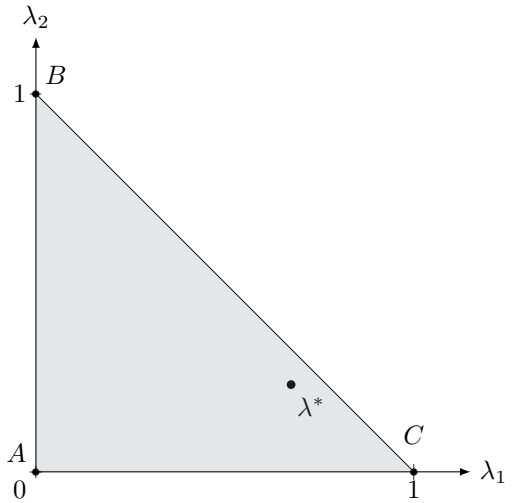


FIGURE 3.4 – Illustration de l'ensemble des paramètres admissibles Λ_Θ initial.

Maintenant, exécutons l'Algorithme 7 avec $\delta = 0$ en 4 étapes :

Étape 1 : *Puisque $E_i = E$ et $MMR(E_i, W) = 2 > 0$, on demande au décideur de comparer deux tâches. Selon la stratégie CSS (Section 1.3.1) employée pour poser des questions, il est intéressant de demander au décideur de comparer les tâches j_1 et j_4. Nous avons $w(j_4) = 7 \geq w(j_1) = 6.7$ par l'équation 3.7 avec $\lambda = \lambda^*$. Par conséquent, la réponse du décideur sera : "la tâche j_4 est plus importante que la tâche j_1". Ensuite, W est mis à jour en imposant la contrainte $w(4) \geq w(1)$, ce qui revient à restreindre Λ_Θ avec l'inégalité $\lambda_2 \geq 2\lambda_1 - 7\lambda_3$. Maintenant, Λ_Θ est représenté par le triangle DCE en Figure 3.5(a). De plus, nous avons maintenant $MMR(E_i, W) = MR(\{j_4\}, E_i, W) = 0$ et donc la tâche j_4 est ajoutée à l'ensemble indépendant en construction (actuellement vide) X.*

Étape 2 : *Nous avons $E_i = E \backslash \{j_4\}$ et $MMR(E_i, W) = MR(\{j_1\}, E_i, W) = 0$. Par conséquent, nous n'avons pas besoin de solliciter le décideur ici. La tâche j_1 est ajoutée à X directement et $X = \{j_4, j_1\}$.*

Étape 3 : *On a $E = E \backslash \{j_4, j_1\}$. Là encore, aucune question n'est nécessaire puisque $MMR(E, W) = MR(\{j_6\}, E_i, W) = 0$. La tâche j_6 est ajoutée à X puisque l'ordonnancement*

(j_6, j_1, j_4) *est réalisable. On a donc* $X = \{j_4, j_1, j_6\}$.

Étape 4 : *On a* $E_i = E \backslash \{j_4, j_1, j_6\}$ *et* $MMR(E_i, W) = 0.67 > 0$. *On demande donc au décideur de comparer les deux tâches* j_3 *et* j_7. *Puisque nous avons* $w(j_3) = 4.3 \geq w(j_7) = 3.6$, *le décideur répond : "La tâche* j_3 *est plus importante que la tâche* j_7*". Ensuite,* W *est mis à jour en imposant la contrainte* $w(3) \geq w(7)$, *ce qui revient à restreindre davantage* Λ_Θ *avec* $\lambda_2 \leq \lambda_1 - \frac{1}{2}\lambda_3$. *Maintenant,* Λ_Θ *est représenté par le polyèdre CEGF Figure 3.5(b). De plus, puisque* $MMR(E_i, W) = MR(\{j_3\}, E_i, W) = 0$, *la tâche* j_3 *est ajoutée à* X *car l'ordonnancement* (j_3, j_6, j_1, j_4) *est réalisable. On obtient* $X = \{j_4, j_1, j_6, j_3\}$. *Enfin, l'algorithme s'arrête puisque* $|X| = r(\mathcal{M}) = 4$ *et on sait que* $X = \{j_4, j_1, j_6, j_3\}$ *est optimal. Après seulement deux questions, nous savons que le jeu de poids du décideur se trouve dans le polyèdre CEGF de la Figure 3.5(b) et cela est suffisant pour déterminer une solution optimale sans aucune ambiguïté.*

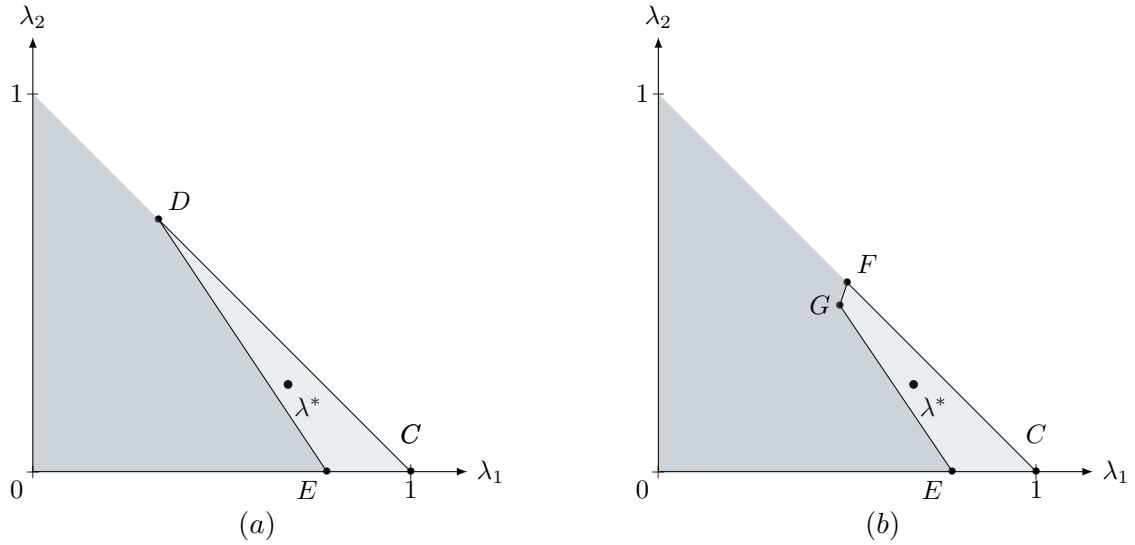


FIGURE 3.5 – Évolution de Λ_Θ au cours de l'Algorithme 7.

Après l'illustration de notre algorithme glouton interactif, l'exemple suivant décrit l'exécution de l'algorithme de recherche locale interactif sur cette même instance.

Exemple 3.4.3. *Considérons le même problème d'ordonnancement que précédemment et exécutons l'algorithme 8 avec* $\delta = 0$. *Supposons que* `GenerationBaseInitiale` $(\mathcal{M})$ *renvoie* $B = \{j_1, j_2, j_4, j_7\}$ *correspondant à l'ordonnancement* (j_2, j_1, j_4, j_7) *dont le vecteur performance est* $y(B) = (19, 23, 22)$.

Étape 1 : *La base* B *n'a que 7 candidats voisins admissibles dont les vecteurs performance sont les suivants :*

	$B_{2,3}$	$B_{2,5}$	$B_{2,6}$	$B_{4,3}$	$B_{4,6}$	$B_{7,3}$	$B_{7,6}$
y_1	22	18	23	16	17	21	22
y_2	21	21	22	18	19	21	22
y_3	20	23	18	26	24	21	19

où $B_{j,j'}$ est la base obtenue en remplaçant la tâche j par la tâche j' dans B. Puisque $MMR(N_B, W) = 4 > 0$, on demande au décideur de comparer les deux bases B et $B_{2,6}$ (toujours selon la CSS). Nous avons $w(B_{2,6}) = 22.2 \geq w(B) = 20.2$ par l'équation (3.7), et donc le décideur répond : "$B_{2,6}$ est meilleure que B". L'algorithme met alors à jour W (correspondant initialement à la Figure 3.4) en imposant la contrainte $w(B_{2,6}) \geq w(B)$, ce qui revient à restreindre Λ_Θ en insérant $\lambda_2 \geq 4\lambda_1 - 4\lambda_3$ (voir Figure 3.6(a) où Λ_Θ est représenté par le triangle CDE). Maintenant, $MMR(N_B, W) = 0.5 > 0$ ce qui nécessite de comparer les deux bases $B_{2,6}$ et $B_{2,3}$. Ici, nous avons $w(B_{2,6}) = 22.2 \geq w(B_{2,3}) = 21.5$. Par conséquent, W est mis à jour en imposant $w(B_{2,6}) \geq w(B_{2,3})$, c'est-à-dire que Λ_Θ est restreint en ajoutant $\lambda_2 \geq 5\lambda_3 - \lambda_1$ (voir la Figure 3.6(b) où Λ_Θ est le polyèdre CDGF). Maintenant l'algorithme arrête de solliciter le décideur puisque $MMR(N_B, W) = MR(B_{2,6}, N_B, W) = 0$. On dispose maintenant d'une nouvelle solution courante $B = B_{2,6} = \{j_6, j_1, j_4, j_7\}$ pour l'étape suivante.

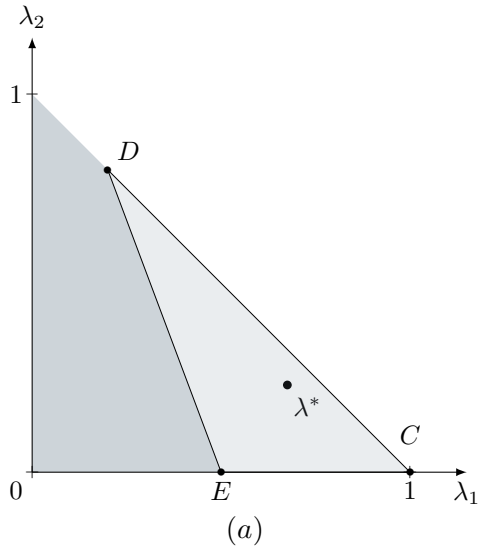


FIGURE 3.6 – Évolution de Λ_Θ au cours de l'exécution de l'Algorithme 8.

Étape 2 : *Ici, N_B ne comprend que 2 bases candidates, à savoir $B = B_{2,6}$ et $B' = \{j_1, j_4, j_6, j_3\}$, correspondant à l'ensemble $\{j_3, j_6, j_1, j_4\}$ dont le vecteur performance est $(25, 20, 17)$. Puisque $MMR(N_B, \Lambda_\Theta) = 1.2 \geq 0$, on demande au décideur de comparer B et B'. Nous avons $w(B') = 23 \geq w(B) = 22.2$ et le décideur répond : "B' est meilleure que B". Alors W est mis à jour en imposant $w(B') \geq w(B)$ et Λ_Θ est restreint par $\lambda_2 \leq \lambda_1 - \frac{1}{2}\lambda_3$, visible en Figure 3.6(c) par le polyèdre CFIH. Maintenant, $MMR(N_B, W) = MR(B', N_B, W) = 0$ et nous fixons $B = B'$ pour l'étape suivante.*

Étape 3 : *Ici,* $MMR(N_B, W) = MR(B, N_B, W) = 0$ *et donc aucune question n'est nécessaire. L'algorithme se termine en retournant la base* B *associée à l'ordonnancement* $\{j_3, j_6, j_1, j_4\}$ *qui est optimal pour le décideur (car on a choisi* $\delta = 0$*).*

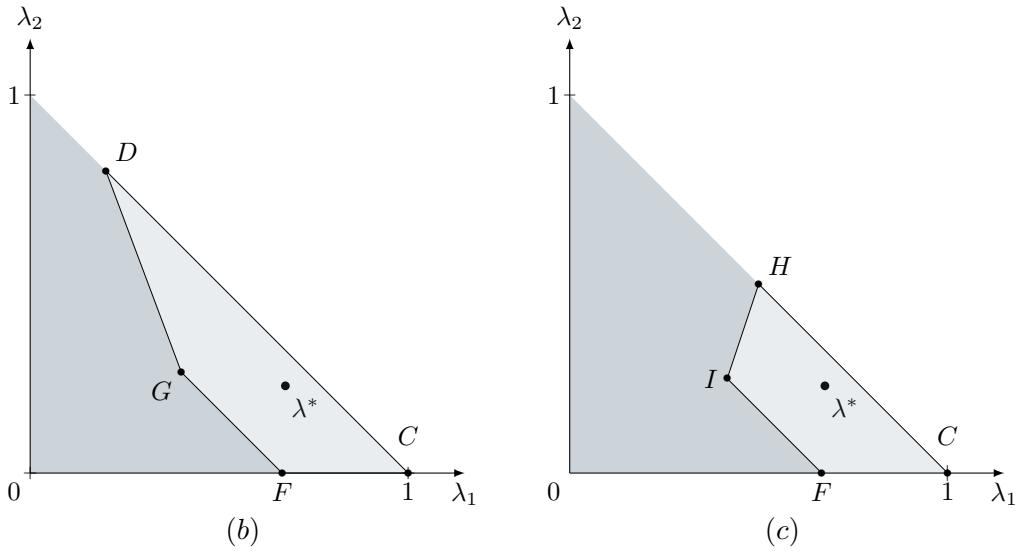


FIGURE 3.6 – Évolution de Λ_Θ au cours de l'exécution de l'Algorithme 8.

Résultats numériques

Dans ce paragraphe, nous évaluons les performances des algorithmes 7 et 8 sur différentes instances du problème d'ordonnancement. Nous considérons des instances impliquant un ensemble de 50 tâches, avec des échéances tirées au hasard entre 1 et 25 de sorte que seuls environ la moitié des tâches puissent être ordonnancées (afin qu'une sélection des tâches soit nécessaire).

		Recherche locale			Glouton		
δ	p	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur(%)
0%	4	4.15	19.9	0.00	4.43	42.6	0.00
	6	8.21	42.7	0.00	8.33	79.9	0.00
	8	21.64	86.1	0.00	14.47	122.9	0.00
20%	4	4.12	12.5	0.02	2.20	14.2	0.00
	6	6.46	23.2	0.03	3.04	27.2	0.00
	8	11.97	38.9	0.01	4.52	41.6	0.00

TABLE 3.2 – Glouton face à la recherche locale sur le matroïde transversal.

On observe dans le Tableau 3.2 que la recherche locale génère moins de questions que l'algorithme glouton, en particulier lorsque $\delta = 0$. Dans ce cas on passe de 123 questions pour 6 critères à 86 avec la recherche locale. Lorsque $\delta = 20\%$ l'écart entre le nombre de questions des deux méthodes diminue, la recherche locale pose 39 questions contre 42 pour le glouton (pour 6 critères). Le seuil δ qui, on le rappelle représente une borne sur le regret final, est ici présenté en pourcentage. En effet, cela correspond à 20% du regret initial.

Malgré 20% d'erreur à l'optimale autorisé, l'erreur empirique reste très faible pour les deux algorithmes, inférieure à 0.03%. On peut constater une performance légèrement meilleure sur ce point pour l'algorithme glouton. On constate également qu'avec $\delta = 20\%$ les deux algorithmes réduisent considérablement le nombre de questions posées au décideur.

Au sujet des temps d'exécution, la recherche locale est un peu plus lente pour $\delta = 0$, mais l'écart de rapidité se creuse avec delta non nul. En effet, l'algorithme glouton est 1.5 fois plus rapide pour 8 critères et $\delta = 0$ que la recherche locale. Lorsque $\delta = 20\%$, glouton devient alors 2.7 fois que rapide que la recherche locale pour 8 critères. Effectivement, les minimax regrets sont calculés sur des sous-ensembles de l'ensemble E, de taille 50, pour la recherche gloutonne, alors qu'ils sont calculés sur des voisinages de taille au plus égale à 300 (car tous les échanges élémentaires sont autorisés) pour la recherche locale. Il est donc assez logique que les temps d'exécution soient plus élevés, et ce même si la condition d'indépendance limite la taille de ce voisinage. Nous verrons dans la suite de cette section que l'impact dépend du type de matroïde. Aussi l'écart de temps d'exécution devient plus important lorsque le nombre d'objectifs augmente, en raison du nombre croissant de voisins non Pareto dominés. En effet, plus il y a de critères, moins il y a tendance à avoir de vecteurs de performances Pareto dominés, car il est plus difficile d'être au moins aussi bon sur chacun des critères.

Ces éléments d'explication peuvent être observés dans les Figures 3.7 et 3.8 où les courbes en trait pointillées représentent les résultats de l'algorithme glouton et les courbes en trait plein ceux de la recherche locale, pour $\delta = 0$. En bleu, le nombre de critères est de 4, 6 en rouge et 8 en vert. On y voit, Figure 3.7, le nombre d'éléments non Pareto-dominés en fonction de l'itération, c'est à dire la taille du voisinage de la base courante B, noté N_B après filtrage par dominance de Pareto pour la recherche locale et la taille de l'ensemble E après le même filtrage pour la méthode gloutonne. Si on regarde les courbes pour 8 critères, à la première itération, il y a environ 4 fois plus d'alternatives présentées pour le calcul du minimax regret avec la recherche locale qu'avec la méthode gloutonne. Cela se répercute sur le nombre de questions, Figure 3.8 adjacente, où 10 questions supplémentaires sont posées au décideur par la recherche locale par rapport au glouton. Dans les résultats finaux, le nombre de questions posées au décideur est pourtant plus bas avec la recherche locale. Deux explications peuvent être données. D'une part, la notion de voisinage de la recherche locale permet d'avoir un nombre d'itérations réduit en comparaison du glouton (6 en moyenne contre 25). De plus, on observe que le nombre de bases non dominées diminue très fortement et rapidement, jusqu'à descendre en dessous du nombre d'éléments non dominés du glouton, dès la troisième itération. En effet, la dominance de Pareto et la vérification d'indépendance permettent une réduction importante du nombre de bases voisines lorsque l'on utilise la recherche locale (cela divise environ par 3 ou plus le nombre de bases voisines).

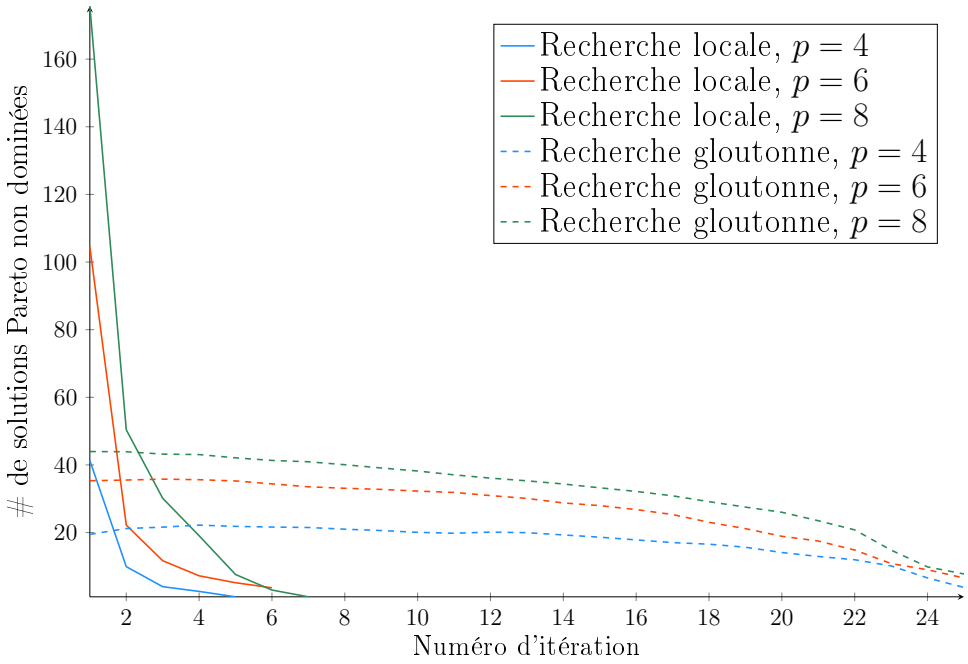


FIGURE 3.7 – Nombre d'alternatives non Pareto dominées en fonction de l'itération sur le matroïde transversal.

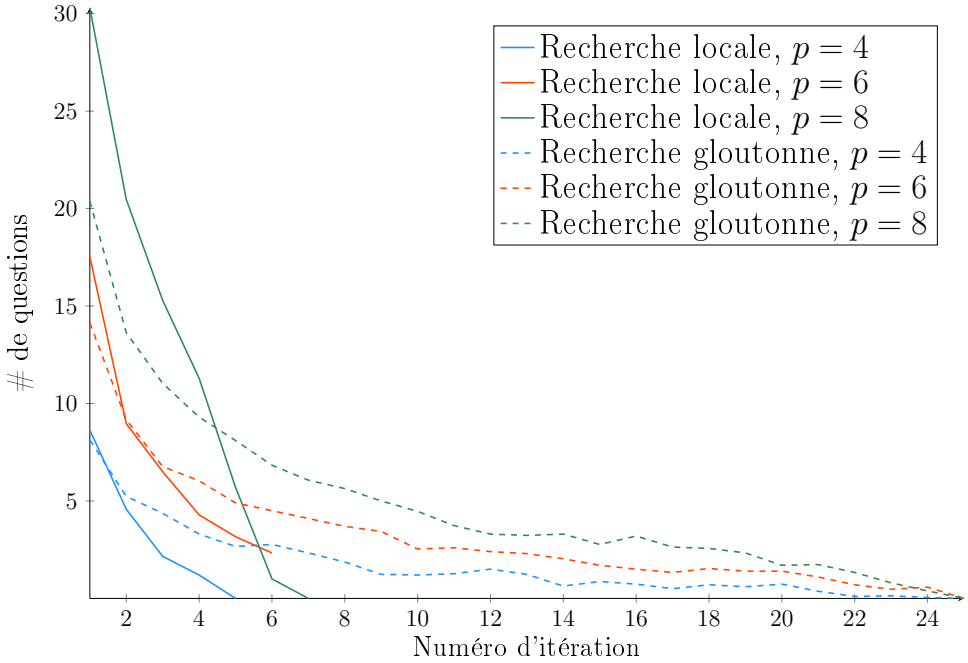


FIGURE 3.8 – Nombre de questions posées au décideur en fonction de l'itération sur le matroïde transversal.

Avec $\delta = 20\%$, on constate une réduction significative du nombre de questions pour les deux algorithmes, qui sont très proches, tout en conservant des solutions de bonne qualité, l'erreur étant au plus égale à 0.02%.

3.4.2 Le matroïde uniforme

Le matroïde uniforme est défini par $\mathcal{I} = \{X \subseteq E : |X| \leq k\}$ pour un entier positif donné $k \leq n$ (défini plus précisément en Section 3.1.1).

Problème de sélection de sous-ensembles. Nous considérons le problème de sélection de sous-ensembles correspondant exactement au matroïde uniforme, c'est-à-dire que l'on cherche à sélectionner k éléments de l'ensemble tout en maximisant la fonction w.

Dans le cadre de ce matroïde, le test d'indépendance pour un ensemble $X \subseteq E$ se fait simplement en vérifiant que la taille de X ne dépasse pas l'entier k fixé.

		Recherche Locale			Glouton		
δ	p	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)
0%	4	3.99	18.5	0.00	5.15	43.9	0.00
	6	13.61	46.2	0.00	10.42	99.8	0.00
	8	45.26	81.0	0.00	13.16	124.4	0.00
20%	4	3.08	13.7	0.01	2.84	17.6	0.00
	6	7.60	28.9	0.01	4.52	37.4	0.00
	8	35.97	36.0	0.02	3.99	38.1	0.01

TABLE 3.3 – Glouton face à la recherche locale sur le matroïde uniforme.

Résultats numériques

Ici, la taille de l'ensemble de base est $|E| = 50$ et $k = 25$, c'est-à-dire que $\mathcal{I} = \{X \subseteq E : |X| \leq 25\}$.

Dans le Tableau 3.3, nous observons que la recherche locale est plus efficace que le glouton en termes de nombre de questions. Par exemple, l'algorithme de recherche locale est environ deux fois plus performant pour 6 critères, soit 46 questions contre 99.

Cependant, l'algorithme glouton est plus rapide que la recherche locale en termes de temps de calcul, 3.5 fois plus rapide avec $\delta = 0$ et 9 fois pour $\delta = 20\%$ non nul. On a vu, avec le matroïde transversal, la raison de cet écart. Cependant, l'amplitude de cet écart devient plus importante avec le matroïde uniforme qu'avec le transversal, pour lequel le glouton était (seulement) au maximum 2.7 fois plus rapide que la recherche locale (avec 8 critères).

Ce problème est très proche du problème d'ordonnancement, mais dans ce dernier le test d'indépendance restreint davantage le nombre de sous-ensembles possibles dans la procédure de recherche locale : un ordonnancement de toutes les tâches de l'ensemble doit exister, or pour le matroïde uniforme, seule la taille est imposée, ce qui explique pourquoi la recherche locale est plus lente ici (il y a plus de bases à comparer, et ainsi le calcul du minimax regret prend plus de temps).

Avec $\delta = 20\%$, de la même manière que pour le matroïde précédent, on constate une réduction significative du nombre de questions, tout en conservant des solutions de bonne qualité (erreur au plus égale à 0.02%).

3.4.3 Le matroïde graphique

Le matroïde graphique se définit de la façon suivante : Étant donné un graphe $G = (V, E)$ où V est l'ensemble des nœuds et E l'ensemble des arêtes, les ensembles indépendants sont les sous-ensembles d'arêtes qui ne contiennent aucun cycle (c'est-à-dire les forêts).

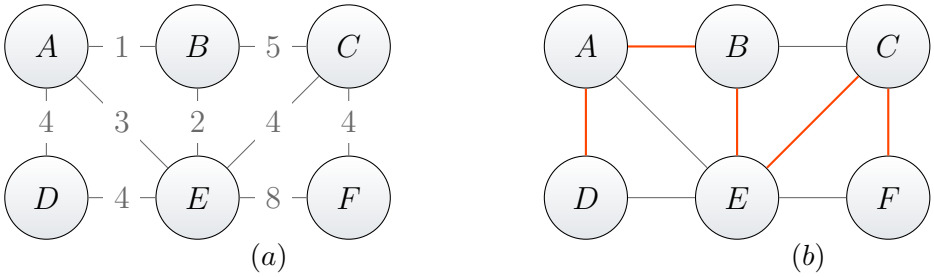


FIGURE 3.9 – Arbre couvrant de poids minimum dans un graphe pondéré.

Problème de l'arbre couvrant minimum. Un arbre couvrant d'un graphe correspond à un arbre inclus dans ce graphe qui connecte tous les sommets entre eux. Ce graphe doit être non orienté et connexe. Le problème d'arbre couvrant de poids minimum impose en plus que les arêtes soient pondérées. Un arbre couvrant de poids minimal d'un tel graphe est un arbre couvrant dont la somme des poids des arêtes est minimale. En Figure 3.9 est présenté un arbre couvrant minimal (cf. Figure 3.9 (b)) d'un graphe pondéré (cf. Figure 3.9 (a)).

Dans le cas du matroïde graphique, toute base correspond à un arbre couvrant du graphe. De plus, si on considère :

— une fonction $c : E \to \mathbb{R}_+$ définissant le coût de chaque arête $e \in E$,
— une constante quelconque strictement supérieure à $\max_{e \in E} c(e)$, notée c^*,
— pour chaque arête $e \in E$, w définie de la façon suivante : $w(e) = c^* - c(e)$,

alors trouver la base optimale revient à résoudre le problème de l'arbre couvrant de coût minimal

Pour ce matroïde, on rappelle que le test d'indépendance d'un ensemble consiste à chercher s'il existe un cycle. Dans le cadre de nos travaux nous avons utilisé la librairie C++

LEMON [1] [Dezső et al., 2011] qui fournit une fonction de détection de cycle pour un graphe G donné. Il est basé sur l'algorithme de parcours en profondeur, appelé DFS (Depth-First Search). L'algorithme réalise un parcours en profondeur mais avec l'adaptation suivante : au développement d'un sommet s, on doit tester s'il existe un arc "retour", c'est-à-dire un arc reliant s à un sommet qui est dans l'ensemble des sommets ouverts.

Résultats numériques.

Ici, les éléments de l'ensemble de base sont les arêtes d'un graphe connexe. Nous avons généré des graphes avec 50 nœuds et une densité égale à 50% (50% des arêtes par rapport au graphe complet sont sélectionnées), toujours grâce à la librairie LEMON. Sinon il suffit de ne placer que la moitié des arêtes, tout en vérifiant que le graphe est connexe.

		Recherche locale			Glouton		
δ	p	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)
0%	4	17.79	41.7	0.00	19.51	64.0	0.00
	6	61.29	105.1	0.00	151.83	154.9	0.00
	8	256.07	195.0	0.00	583.93	262.3	0.00
0.2	4	16.82	13.2	0.14	10.12	13.4	0.28
	6	33.01	24.7	0.17	40.92	26.2	0.24
	8	68.88	37.0	0.25	99.20	40.9	0.28

TABLE 3.4 – Glouton face à la recherche locale sur le matroïde graphique.

Dans le Tableau 3.4, nous observons que la recherche locale est plus rapide que la recherche gloutonne, à l'inverse des observations faites sur le matroïde uniforme et sur le matroïde transversal. Ceci est principalement dû à une plus grande taille de l'ensemble de base. En effet, un graphe complet compte $\frac{|E|\times(|E|-1)}{2}$ arêtes mais, dans le cadre de ces travaux, on travaille sur des graphes de densité égale à 50%. Conséquemment, le nombre d'arêtes est d'environ 600 ici.

Si cela explique la différence majeure de temps d'exécution cela n'explique pas entièrement le fait que la recherche locale soit plus rapide que la recherche gloutonne. Cette dernière construit la base optimale à partir de l'ensemble vide, alors que la recherche locale part d'une solution complète (obtenue avec `GenerationBaseInitiale`). On peut observer l'effet produit en Figure 3.10. Elle représente le nombre de solutions non Pareto dominées en fonction du nombre d'itérations, permettant d'observer les différences de comportement entre les deux méthodes avec $\delta = 0$. A l'instar des figures présentées pour le matroïde transversal (Figure 3.7 et 3.8), les trois courbes en trait pointillée représentent les résultats de la recherche gloutonne et les courbes en trait plein représentent ceux de la recherche locale (en bleu, résultats pour 4, 6 en rouge et 8 en vert). On remarque que le comportement de chaque

1. https://lemon.cs.elte.hu/trac/lemon

méthode est similaire peu importe le nombre de critères. La réaction est aussi similaire avec les trois matroïdes présentés (uniforme, transversal et graphique). La recherche locale débute avec un nombre important d'alternatives possibles à l'itération 0 alors que la recherche gloutonne se voit proposée légèrement moins de possibilités, environ 1.25 fois moins pour 8 critères, par exemple. Cependant, cet écart était beaucoup plus important avec le matroïde transversal (un rapport de 4). Dès la seconde itération, le glouton se voit présenter un nombre d'alternatives beaucoup plus important par rapport à la recherche locale, ce qui explique les temps d'exécution plus grands.

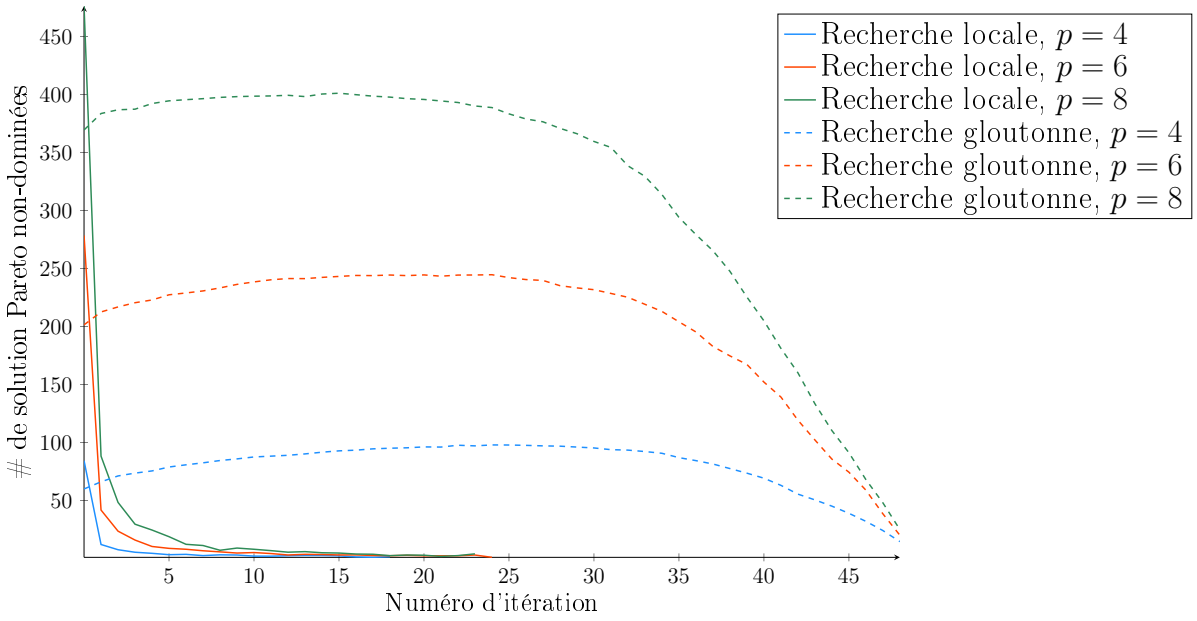


FIGURE 3.10 – Nombre d'alternatives non Pareto dominées en fonction de l'itération sur le matroïde graphique.

Pour l'algorithme glouton, le nombre de solutions Pareto non dominées en fonction de l'itération a une forme presque concave, avec des valeurs très supérieures à celles de la recherche locale. L'effet est très amplifié en comparaison des courbes présentées pour le matroïde transversal. Au lancement de la procédure, la sélection d'un élément de l'ensemble permet à quelques alternatives de devenir non Pareto-dominées. Cet effet s'inverse tardivement, aux alentours de l'itération 25, avec la construction de la solution, qui limite le nombre de non Pareto-dominées à l'instar de la recherche locale qui y parvient beaucoup plus rapidement (notamment grâce à la difficulté d'être indépendant pour un ensemble voisin). Par conséquent, le nombre de questions posées au décideur est drastiquement inférieur avec la recherche locale.

Avec $\delta = 20\%$, on constate une plus grande réduction du nombre de questions pour les deux algorithmes par rapport aux autres matroïdes étudiés. Là où avec la recherche locale et le matroïde transversal on divisait par 2.2 le nombre de questions avec delta non nul, ici on divise par 5.2 ce nombre. Pareillement avec le glouton qui passe de 3 fois moins de question

à 6.3 fois moins. Les résultats restent proches, et conservent des solutions de bonne qualité. L'erreur est inférieure à 0.30%, ce qui est cependant supérieure à la qualité obtenue avec les deux autres matroïdes.

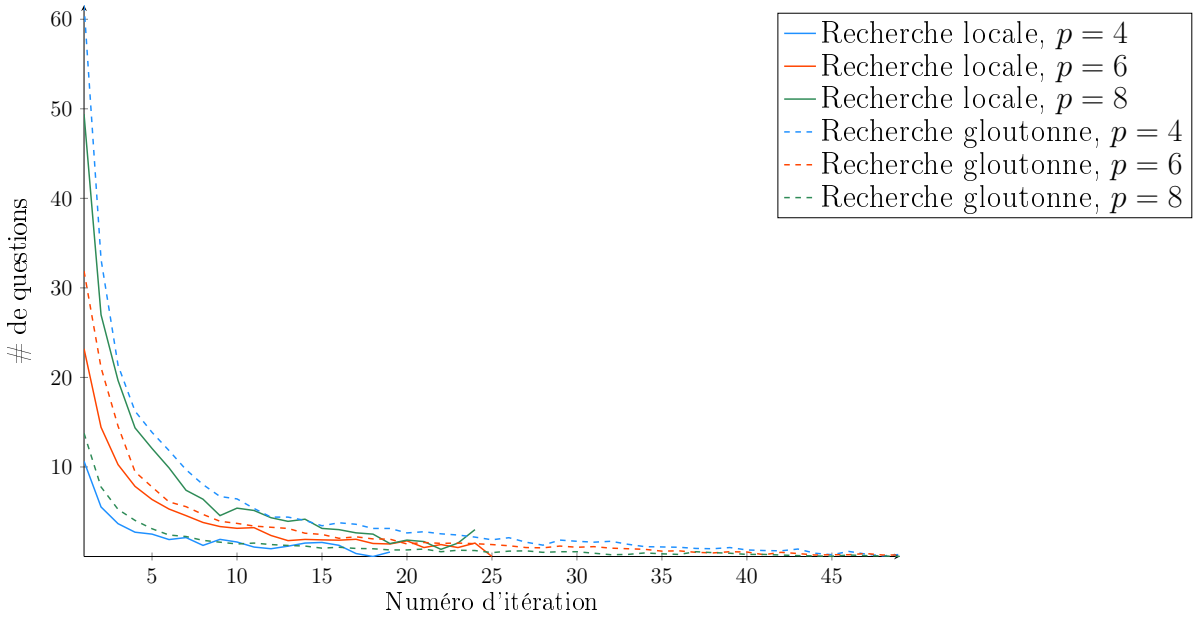


FIGURE 3.11 – Nombre de questions posées au décideur en fonction de l'itération sur le matroïde graphique.

3.5 Conclusion du chapitre

Nous avons introduit deux méthodes d'optimisation interactives combinant l'élicitation des préférences avec l'exploration d'ensembles indépendants dans un matroïde, l'une basée sur la recherche locale et l'autre sur la recherche gloutonne. Le principe commun aux deux méthodes est d'intercaler des questions de préférences avec des étapes d'optimisation afin de concentrer l'effort d'élicitation sur l'obtention des informations préférentielles nécessaires à la détermination d'une solution optimale, ou d'une solution quasi-optimale si l'on veut économiser des questions. L'intérêt de notre proposition est qu'elle est assez générale et peut être mise en œuvre dans divers problèmes d'optimisation impliquant une structure de matroïde. D'autres problèmes pourraient être résolus de manière similaire, comme l'élection de comités avec des contraintes de représentation, la conception de réseaux de capteurs, certains problèmes d'affectation et de transport. Nous avons implémenté ces méthodes sur trois problèmes (ordonnancement, sélection de sous-ensembles et arbre couvrant) et présenté des tests numériques montrant l'efficacité pratique de nos algorithmes, en termes de temps de calcul, de nombre de questions de préférences et d'erreur empirique.

Dans le chapitre suivant, nous avons choisi d'étendre notre approche aux fonctions d'ensemble non linéaires. Par exemple, les fonctions d'ensemble sous-modulaires. Les fonctions

d'utilité sous-modulaires sont également utilisées pour garantir des rendements décroissants stipulant que l'ajout d'un élément à un ensemble plus petit a plus de valeur que son ajout à un ensemble plus grand. De plus, la sous-modularité présente un intérêt particulier dans les problèmes de maximisation sur des domaines combinatoires car elle joue un rôle analogue à la concavité dans l'optimisation continue.

Chapitre 4

Optimisation interactive sous contrainte de matroïde d'une fonction sous-modulaire

Résumé

Pareillement au chapitre précédent, dans ce chapitre nous proposons deux méthodes d'élicitation incrémentale sur des structures de matroïdes pondérées. Lorsque la fonction d'ensemble à optimiser est additive, comme vu précédemment, ce problème peut être résolu de manière exacte en utilisant un algorithme glouton ou de recherche locale. Nous avons montré que cela est aussi le cas dans les situations où la fonction d'ensemble n'est pas exactement connue et doit être obtenue pendant le processus d'optimisation.

Cependant, la fonction d'ensemble n'est pas toujours additive en raison des interactions possibles entre les éléments de l'ensemble. Dans un premier temps, nous considérons le problème de la maximisation d'une fonction d'ensemble sous-modulaire, sous une contrainte de matroïde. Nous proposons deux approches interactives visant à entrelacer l'élicitation de la fonction d'ensemble sous-modulaire avec la construction d'un sous-ensemble indépendant optimal soumis aux contraintes de matroïde. La première est basée sur un algorithme glouton et l'autre sur la recherche locale. Ces algorithmes sont testés sur des problèmes impliquant une structure matroïde et une fonction sous-modulaire à maximiser. Ces résultats ont fait l'objet d'une publication à la conférence Algorithmic Decision Theory de 2021 [Benabbou et al., 2021b]. Dans un second temps, nous considérons le problème de la minimisation d'une fonction d'ensemble sous-modulaire, sous une contrainte de matroïde.

4.1 Adaptation des méthodes précédentes

On a vu que l'optimisation d'une fonction d'ensemble sous une contrainte de matroïde a reçu beaucoup d'attentions depuis les travaux d'Edmonds [Edmonds, 1971]. En théorie de la décision, l'additivité des utilités est souvent remplacée par des fonctions d'utilité sous-modulaires, fréquemment utilisées pour garantir un principe de rendements décroissants [Ahmed and Atamtürk, 2011, Lehmann et al., 2006, Vondrák, 2008]. Dans le cas sous-modulaire, l'ajout d'un élément à un ensemble de plus petite taille a plus de valeur que son ajout à un ensemble plus grand.

Bien que l'on sache que la résolution d'un problème de minimisation d'une fonction sous-modulaire sans contrainte est polynomiale [Schrijver, 2000], la maximisation d'une fonction sous-modulaire est difficile en général car ce problème inclut le problème de la recherche d'une coupe de poids maximum dans un graphe (max-cut). Des algorithmes de recherche locale et par approche gloutonne ont été proposés pour le problème en maximisation et certaines limites intéressantes sur la qualité des approximations retournées sont connues [Calinescu et al., 2011, Nemhauser et al., 1978, Skowron, 2017, Vondrák, 2008]. Dans cette section, nous nous concentrons sur le problème de recherche d'un maximum global d'une fonction sous-modulaire et monotone sous une contrainte de matroïde. Plus précisément, nous proposons deux algorithmes qui étendent les algorithmes proposés dans le chapitre précédent pour les fonctions d'ensemble additives [Benabbou et al., 2021a]. On rappelle brièvement que pour un matroïde $\mathcal{M}(E, \mathcal{I})$, w est une fonction d'ensemble définie sur 2^E, monotone par rapport à l'inclusion, et mesurant le poids de tout sous-ensemble de E, avec $w(\emptyset) = 0$ (voir Section 3.1.1 pour plus de détails).

Un algorithme glouton interactif

Pour les problèmes où w est exactement observable, de bonnes solutions approximatives peuvent être construites à l'aide de l'algorithme glouton simple suivant : en partant de $X = \emptyset$, l'idée est de sélectionner un élément $e \in E \backslash X$, sans perdre la propriété d'indépendance, qui maximise la contribution marginale de X, c'est-à-dire

$$\Delta(e|X) = w(X \cup \{e\}) - w(X) \tag{4.1}$$

L'algorithme s'arrête lorsque plus aucun élément ne peut être ajouté à X. L'ensemble X est une base à la fin de la procédure. Quand w est additif, cela revient à appliquer l'algorithme glouton de la Section 3.1.3. Pour les fonctions monotones sous-modulaires, cet algorithme glouton a un taux d'approximation de $(1 - \frac{1}{e}) \approx 0.63$ pour le matroïde uniforme et un taux d'approximation de $\frac{1}{2}$ dans le cas général [Fisher et al., 1978, Nemhauser et al., 1978].

Pour les problèmes où la fonction d'ensemble w est connue de manière imprécise, nous proposons une version interactive de cet algorithme glouton classique qui génère des questions de préférences uniquement lorsqu'il est nécessaire de savoir comparer certains éléments. Plus précisément, les questions sont générées uniquement lorsque les données de préférences disponibles ne sont pas suffisantes pour identifier un élément qui pourrait être ajouté à l'ensemble X de manière à avoir une bonne solution approximative avec garanties de performances.

Nous mettons en œuvre cette idée en calculant les minimax regrets sur l'ensemble $\mathcal{S} = \{X \cup \{e\} : e \in E \backslash X \text{ et } X \cup \{e\} \in \mathcal{I}\}$. Plus précisément, pour savoir quel élément ajouter à X à l'étape i de l'algorithme glouton, on pose des questions de préférences jusqu'à ce que $MMR(\mathcal{S}, W)$ tombe en dessous d'un seuil donné $\delta_i \geq 0$, où δ_i est une fraction du seuil de tolérance δ telle que $\sum_{i=1}^{r(\mathcal{M})} \delta_i = \delta$ (voir Algorithme 9). On considère une version interactive où, à chaque étape, on demande au décideur de comparer des ensembles de la forme $X \cup \{e\}$ avec $e \in E \backslash X$ et d'indiquer lequel il préfère. Dans les tests nous verrons que lorsque w est une fonction paramétrée, on peut économiser beaucoup de questions de préférences.

Algorithme 9 Algorithme glouton interactif

$X \leftarrow \emptyset$; $E_c \leftarrow E$
for $i = 1 \dots r(\mathcal{M})$ **do**
  $\mathcal{S} \leftarrow \{X \cup \{e\} : e \in E_c\}$
  while $\text{MMR}(\mathcal{S}, W) > \delta_i$ **do**
    Demander au décideur de comparer deux éléments de $\mathcal{S}$
    Mettre à jour W selon la réponse du décideur
  end while
  Sélectionner $e \in E_c$ tel que $\text{MR}(X \cup \{e\}, \mathcal{S}, W) \leq \delta_i$ et déplacer e de E_c à X
  Supprimer de E_c tout élément e tel que $X \cup \{e\} \notin \mathcal{I}$
end for
return X

Notons que l'Algorithme 9 ne génère pas plus qu'un nombre polynomial de questions au décideur. À chaque étape, le nombre de questions est en effet limité par $|E|^2$ car les comparaisons sont générées jusqu'à ce que $MMR(\mathcal{S}, W) \leq \delta_i$, où $\mathcal{S} \subseteq \{X \cup \{e\} : e \in E\}$. Dans le pire des cas, on demande au décideur de comparer tous les éléments de $\mathcal{S}$. Par conséquent, le nombre d'opération de la boucle "while" est également polynomial. On note cependant que l'implémentation peut différer significativement d'un contexte d'application à un autre. De la même façon que dans le chapitre précédent, vérifier si $X \cup \{e\} \in \mathcal{I}$ peut être plus ou moins complexe selon le matroïde considéré. Par exemple, en considérant les matroïdes uniforme et de partition, les tests d'indépendance peuvent être effectués en temps polynomial.

Une deuxième source de complexité est le calcul des valeurs MMR, qui peut être plus ou moins simple selon les hypothèses faites sur w. Une option intéressante est de se concentrer sur les fonctions paramétriques qui sont linéaires dans leurs paramètres (par exemple, une combinaison linéaire de fonctions splines, ou une utilité multicritère linéaire, ou une moyenne pondérée ordonnée de valeurs de critères). Dans ce cas, l'optimisation des regrets peut être effectuée en temps polynomial en utilisant la programmation linéaire. De plus, définir w par une fonction paramétrique permet de réduire le nombre de questions en pratique, puisque toute déclaration de préférence du type $w(X) \geq w(X')$ se traduit par une contrainte sur l'espace des paramètres, réduisant les préférences possibles sur d'autres sous-ensembles (comme nous le verrons dans les Sections 4.3 et 4.4).

Un algorithme de recherche locale interactif

Dans ce paragraphe, nous nous concentrons sur l'approche de recherche locale déjà décrite dans le chapitre précédent : en partant d'une base arbitraire X, l'idée est de remplacer un élément $e \in X$ par un élément $e' \in E \backslash X$ tel que $X \cup \{e'\} \backslash \{e\}$ appartient à $\mathcal{I}$ et est meilleur que X. Ce principe d'échange simple peut être itéré jusqu'à atteindre un optimum local.

Lorsque w est exactement observable et avec une fonction d'ensemble sous-modulaire, l'algorithme de recherche locale a un taux d'approximation de 1/2, même dans le cas particulier d'un matroïde uniforme [Fisher et al., 1978]. Lorsque w n'est pas connu, l'algorithme de recherche locale peut être combiné avec une méthode d'élicitation des préférences qui recueille des données sur les préférences uniquement lorsque cela est nécessaire pour identifier un échange améliorant. Pour mettre en œuvre cette idée, il suffit d'utiliser l'algorithme 8 déjà utilisé pour les fonctions d'ensemble additives.

4.2 Garantie de performance sur la qualité de la solution retournée

Cette section est consacrée aux garanties théoriques sur la qualité de la solution retournée par les deux algorithmes présentés dans la section précédente.

4.2.1 Cas de la méthode gloutonne

Avant de considérer le cas général, concentrons-nous sur le matroïde uniforme.

Proposition 4.2.1. *Soit W_f l'ensemble final W lorsque l'Algorithme 9 s'arrête. Pour le matroïde uniforme, l'Algorithme 9 garantit de retourner une base X telle que :*

$$\forall w \in W_f,\ w(X) \geq \big(1 - \frac{1}{e}\big) w(X^*) - \delta, \quad \text{où } X^* \in \arg\max_{Y \in \mathcal{I}} w(Y).$$

Démonstration. Soit $w \in W_f$ et soit $X^* \in \arg\max_{Y \in \mathcal{I}} w(Y)$. Nous voulons prouver que $w(X) \geq \big(1 - \frac{1}{e}\big) w(X^*) - \delta$. Soit e_i, $i \in \{1, \ldots, r(\mathcal{M})\}$, le $i^{\text{ème}}$ élément inséré dans la solution en construction X pendant l'exécution de l'algorithme 9. Soit X_i l'ensemble X à la fin de la $i^{\text{ème}}$ itération (c'est-à-dire $X_i = \{e_1, \ldots, e_i\}$). Soit W_i (resp. $\mathcal{S}_i$) l'ensemble d'incertitude W (resp. l'ensemble $\mathcal{S}$) à la fin de la $i^{\text{ème}}$ étape d'itération. Soit e_i^*, $i \in \{1, \ldots, r(\mathcal{M})\}$, le $i$ème élément de X^* dans un ordre arbitraire.

Pour toute étape $i \in \{1, \ldots, r(\mathcal{M})\}$, nous avons $MR(X_{i-1} \cup \{e_i\}, \mathcal{S}_i, W_i) \leq \delta_i$ dû à la ligne 9. Puisque $w \in W_f \subseteq W_i$, nous savons que $w(X_{i-1} \cup \{e\}) - w(X_{i-1} \cup \{e_i\}) \leq \delta_i$ pour tout $e \in E_c$, où $E_c = E \backslash X_{i-1}$ pour le matroïde uniforme (voir lignes 2 et 10). Ensuite, on peut déduire de l'équation (4.1) que $\Delta(e|X_{i-1}) - \Delta(e_i|X_{i-1}) \leq \delta_i$ pour tous les $e \in E \backslash X_{i-1}$. La dernière inégalité est également valable pour tout $e \in X_{i-1}$ car $\Delta(e|X_{i-1}) = 0$. Par conséquent, pour toute étape $i \in \{1, \ldots, r(\mathcal{M})\}$, nous avons :

$$\Delta(e|X_{i-1}) - \Delta(e_i|X_{i-1}) \leq \delta_i,\ \forall e \in E \tag{4.2}$$

On obtient alors :

$$\begin{aligned}
w(X^*) &\leq w(X_{i-1} \cup X^*) \text{ (puisque } w \text{ est monotone)} \\
&= w(X_{i-1}) + \sum_{j=1}^{r(\mathcal{M})} \big(w(X_{i-1} \cup \{e_1^*, \ldots, e_j^*\}) - w(X_{i-1} \cup \{e_1^*, \ldots, e_{j-1}^*\})\big) \\
&= w(X_{i-1}) + \sum_{j=1}^{r(\mathcal{M})} \Delta(e_j^* | X_{i-1} \cup \{e_1^*, e_2^*, \ldots, e_{j-1}^*\}) \text{ (par l'équation (4.1))} \\
&\leq w(X_{i-1}) + \sum_{j=1}^{r(\mathcal{M})} \Delta(e_j^* | X_{i-1}) \text{ (puisque } w \text{ est sous-modulaire)} \\
&\leq w(X_{i-1}) + \sum_{j=1}^{r(\mathcal{M})} (\Delta(e_i | X_{i-1}) + \delta_i) \text{ (par l'équation (4.2))} \\
&= w(X_{i-1}) + r(\mathcal{M}) \times \big(\Delta(e_i | X_{i-1}) + \delta_i\big)
\end{aligned}$$

De la dernière inégalité, nous pouvons déterminer que :

$$\frac{1}{r(\mathcal{M})}\big(w(X^*) - w(X_{i-1})\big) - \delta_i \leq \Delta(e_i | X_{i-1})$$

Ce qui peut se réécrire de la façon suivante :

$$\frac{1}{r(\mathcal{M})}\big(w(X^*) - w(X_{i-1})\big) - \delta_i \leq w(X^*) - w(X_{i-1}) - \big(w(X^*) - w(X_i)\big)$$

puisque $X_i = X_{i-1} \cup \{e_i\}$. Par conséquent, nous avons $\frac{\Pi_{i-1}}{r(\mathcal{M})} - \delta_i \leq \Pi_{i-1} - \Pi_i$ ou de manière équivalente :

$$\Pi_i \leq \big(1 - \frac{1}{r(\mathcal{M})}\big)\Pi_{i-1} + \delta_i$$

où Π_i est simplement défini par $\Pi_i = w(X^*) - w(X_i)$ pour tout $i \in \{0, \ldots, r(\mathcal{M})\}$.
En appliquant récursivement cette inégalité, on obtient :

$$\Pi_{r(\mathcal{M})} \leq \left(1 - \frac{1}{r(\mathcal{M})}\right)^{r(\mathcal{M})} \times \Pi_0 + \sum_{i=1}^{r(\mathcal{M})} \delta_i \left(1 - \frac{1}{r(\mathcal{M})}\right)^{r(\mathcal{M})-i}$$

Puis, puisque $\Pi_0 = w(X^*)$ et $\Pi_{r(\mathcal{M})} = w(X^*) - w(X)$, on obtient :

$$w(X^*) - w(X) \leq \left(1 - \frac{1}{r(\mathcal{M})}\right)^{r(\mathcal{M})} \times w(X^*) + \sum_{i=1}^{r(\mathcal{M})} \delta_i \left(1 - \frac{1}{r(\mathcal{M})}\right)^{r(\mathcal{M})-i}$$

ou de manière équivalente :

$$w(X) \geq \left(1 - \big(1 - \frac{1}{r(\mathcal{M})}\big)^{r(\mathcal{M})}\right) w(X^*) - \sum_{i=1}^{r(\mathcal{M})} \delta_i \left(1 - \frac{1}{r(\mathcal{M})}\right)^{r(\mathcal{M})-i}$$

Enfin, en utilisant $1 - x \leq e^{-x}$ pour tout $x \in \mathbb{R}$, et $1 - \frac{1}{x} \leq 1$ pour tout $x \in \mathbb{R}_+^*$, on obtient :

$$w(X) \geq \big(1 - \frac{1}{e}\big) w(X^*) - \sum_{i=1}^{r(\mathcal{M})} \delta_i = \big(1 - \frac{1}{e}\big) w(X^*) - \delta$$

□

Notons que la proposition 4.2.1 ne peut pas être étendue dans le cas d'un matroïde général car l'équation 4.1 n'est plus valable puisque $E_c \neq E \backslash X_i$ dans le cas général et on peut avoir des éléments $e \in E$ ne vérifiant pas la propriété d'indépendance mais donnant une meilleure contribution marginale.
Nous établissons maintenant un résultat plus général.

Proposition 4.2.2. *Soit W_f l'ensemble final W lorsque l'algorithme 9 s'arrête. L'algorithme 9 garantit de retourner une base X telle que :*

$$\forall w \in W_f,\ w(X) \geq \frac{1}{2}\big(w(X^*) - \delta\big), \quad \textit{où } X^* \in \arg\max_{Y \in \mathcal{I}} w(Y).$$

Démonstration. Soit $w \in W_f$ et soit $X^* \in \arg\max_{Y \in \mathcal{I}} w(Y)$. Nous voulons prouver que $w(X) \geq \frac{1}{2}(w(X^*) - \delta)$. Soit e_i, $i \in \{1, \ldots, r(\mathcal{M})\}$, le $i^{\text{ème}}$ élément inséré dans X pendant l'exécution de l'algorithme 9. Soit X_i l'ensemble X à la fin de la $i^{\text{ème}}$ itération (c'est-à-dire $X_i = \{e_1, \ldots, e_i\}$). Soit W_i (resp. $\mathcal{S}_i$) l'ensemble d'incertitude W (resp. l'ensemble $\mathcal{S}$) à la fin de la $i^{\text{ème}}$ étape d'itération. Soit W_i l'ensemble d'incertitude W à la fin de la $i^{\text{ème}}$ itération.

En raison du théorème des échanges multiples (déjà utilisé dans la Section 3.4.2), il existe une correspondance 1-1 $\sigma : X \to X^*$ telle que $B_i = (X \backslash \{e_i\}) \cup \{\sigma(e_i)\}$ est une base du matroïde pour chaque élément $e_i \in X$. Nous pouvons alors en déduire $X_{i-1} \cup \{\sigma(e_i)\} \in \mathcal{I}$ de $X_{i-1} \cup \{\sigma(e_i)\} \subseteq B_i$ (en utilisant l'axiome A_1) et donc, on a nécessairement $X_{i-1} \cup \{\sigma(e_i)\} \in \mathcal{S}_i$ à l'étape i. Puisque $MR(X_{i-1} \cup \{e_i\}, \mathcal{S}_i, W_i) \leq \delta_i$ (ligne 9), on obtient $w(X_{i-1} \cup \{\sigma(e_i)\}) - w(X_{i-1} \cup \{e_i\}) \leq \delta_i$, ce qui peut être réécrit :

$$\Delta(\sigma(e_i)|X_{i-1}) - \Delta(e_i|X_{i-1}) \leq \delta_i \tag{4.3}$$

On obtient alors :

$$\begin{aligned}
w(X^*) &\leq w(X \cup X^*) \text{ (puisque } w \text{ est monotone)} \\
&= w(X) + \sum_{i=1}^{r(\mathcal{M})} \big(w(X \cup \{\sigma(e_1), \ldots, \sigma(e_i)\}) - w(X \cup \{\sigma(e_1), \ldots, \sigma(e_{i-1})\})\big) \\
&= w(X) + \sum_{i=1}^{r(\mathcal{M})} \Delta(\sigma(e_i)|X \cup \{\sigma(e_1), \ldots, \sigma(e_{i-1})\}) \text{ (par l'équation (4.1)))} \\
&\leq w(X) + \sum_{i=1}^{r(\mathcal{M})} \Delta(\sigma(e_i)|X_{i-1}) \text{ (puisque } w \text{ est sous-modulaire et } X_{i-1} \subseteq X)
\end{aligned}$$

$$\leq w(X) + \sum_{i=1}^{r(\mathcal{M})} (\Delta(e_i|X_{i-1}) + \delta_i) \text{ (par l'équation (4.3))}$$
$$= 2w(X) + \sum_{i=1}^{r(\mathcal{M})} \delta_i \text{ (par l'équation (4.1))}$$
$$= 2w(X) + \delta \text{ (ce qui établit le résultat)}$$

□

4.2.2 Cas de la recherche locale

La proposition suivante suggère que la base retournée par l'algorithme 8 est une bonne solution approximative.

Proposition 4.2.3. *Soit W_f l'ensemble final W lorsque l'Algorithme 8 s'arrête. L'algorithme 8 garantit de retourner une base X telle que :*

$$\forall w \in W_f,\ w(X) \geq \frac{1}{2}\big(w(X^*) - \delta\big), \quad \text{où } X^* \in \arg\max_{Y \in \mathcal{I}} w(Y).$$

Démonstration. Soit $w \in W_f$ et soit $X^* \in \arg\max_{Y \in \mathcal{I}} w(Y)$. Nous voulons prouver que $w(X) \geq \frac{1}{2}(w(X^*) - \delta)$. Soit e_i, $i \in \{1, \ldots, r(\mathcal{M})\}$, le $i^{\text{ème}}$ élément de X dans un ordre arbitraire. Soit X_i l'ensemble défini par $X_i = \{e_1, \ldots, e_i\}$. En raison du théorème des échanges multiples, il existe une bijection $\sigma : X \to X^*$ telle que $B_i = (X \backslash \{e_i\}) \cup \{\sigma(e_i)\}$ est une base du matroïde pour chaque élément $e_i \in X$. Notons que $B_i \in N_X$ (le voisinage de X) pour tout $i \in \{1, \ldots, r(\mathcal{M})\}$ puisque B_i diffère de X par exactement un élément. De plus, nous avons $MR(X, N_X, W) \leq \delta / r(\mathcal{M})$ à la fin de l'exécution (grâce à la ligne 10). Par conséquent, $w(B_i) - w(X) \leq \delta / r(\mathcal{M})$ est vérifiée par définition des max regrets, ce qui peut être réécrit de la manière suivante :

$$\Delta(\sigma(e_i)|X \backslash \{e_i\}) - \Delta(e_i|X \backslash \{e_i\}) \leq \frac{\delta}{r(\mathcal{M})} \tag{4.4}$$

en utilisant l'équation (4.1). On obtient alors :

$$w(X^*) \leq w(X) + \sum_{e \in X^*} \Delta(e|X) \text{ (par sous-modularité, voir [Nemhauser et al., 1978] pour une preuve))}$$
$$= w(X) + \sum_{i=1}^{r(\mathcal{M})} \Delta(\sigma(e_i)|X)$$
$$\leq w(X) + \sum_{i=1}^{r(\mathcal{M})} \Delta(\sigma(e_i)|X \setminus \{e_i\}) \text{ (par sous-modularité)}$$
$$\leq w(X) + \sum_{i=1}^{r(\mathcal{M})} \big(\Delta(e_i|X \setminus \{e_i\}) + \frac{\delta}{r(\mathcal{M})}\big) \text{ (par l'équation (4.4))}$$

$$\leq w(X) + \sum_{i=1}^{r(\mathcal{M})} \left(\Delta(e_i|X_{i-1}) + \frac{\delta}{r(\mathcal{M})}\right) \text{ (puisuqe } X_{i-1} \subseteq X\backslash\{e_i\})$$
$$= 2w(X) + \delta \text{ (qui établit le résultat)}$$

□

Malheureusement, nous ne pouvons pas prouver que l'algorithme de recherche locale génère un nombre polynomial de questions et se termine après un nombre polynomial d'itérations comme cela est fait pour l'algorithme 9. Lorsque $\delta \neq 0$, certains cycles de type $(X_1, \ldots, X_t)$ avec $X_{i+1} \in N_{X_i}$ et $X_1 = X_t$ peuvent même se produire. Si un cycle est détecté, il peut être facilement brisé en divisant itérativement δ par deux (comme dans la section précédente). Malgré ces mauvaises propriétés théoriques, nous verrons dans la section expérimentale que l'algorithme de recherche locale obtient de bons résultats en pratique.

4.3 Application au problème de couverture maximale

Le problème de couverture maximale (MCP) consiste à sélectionner au plus k sous-ensembles de manière à couvrir un nombre maximal d'éléments (chaque sous-ensemble couvre un certain nombre d'éléments, c'est-à-dire contient ces éléments). C'est une variante du problème de couverture par ensembles, qui lui consiste à couvrir tous les éléments avec le moins d'ensemble possible.
Plusieurs variantes du MCP ont été étudiées dans la littérature, certaines parmi les plus connues sont présentées ci-dessous :

- MCP sous contrainte de matroïde : dans le problème initial, il n'y a aucune restriction sur l'ensemble que nous pouvons choisir en fonction de ce qui a déjà été choisi. Cependant, dans de nombreuses applications, une telle contrainte est nécessaire, car lorsque nous choisissons un ensemble, on peut avoir une restriction sur le choix des autres ensembles [Filmus and Ward, 2012].
- MCP pondéré : dans cette version, chaque élément a un poids et l'objectif devient alors la maximisation de la somme des poids des éléments couverts [Hochbaum and Pathria, 1998].
- MCP multicritère : chaque élément possède non pas un poids unique mais un vecteur performance établi sur différents critères.
- MCP budgétisé : dans ce cas, chaque élément dispose non seulement d'un poids mais aussi d'un coût et la contrainte porte sur le coût des éléments choisis. Ainsi, l'objectif reste le même que dans la version pondérée, mais s'ajoute une contrainte qui stipule que le coût total (la somme de tous les coûts des éléments couverts) ne soit dépasser un seuil prédéfini (le budget) [Khuller et al., 1999].
- MCP généralisé : dans ce cas le poids d'un élément n'est pas prédéterminé, mais dépend de l'ensemble qui le couvre. Le but reste le même que dans la version budgétisée, soit maximiser le poids total des éléments couverts, tout en maintenant le coût inférieur au budget. Une contrainte supplémentaire dans cette version est que chaque élément

ne peut être couvert que par un seul ensemble. Cela évite toute ambiguïté quant à la détermination du poids d'un élément couvert, au cas où il serait couvert par plusieurs ensembles [Cohen and Katzir, 2008].

Dans la section suivante, nous présentons plus formellement le MCP dans sa variante multicritère sous contrainte de matroïde.

4.3.1 Formalisation de la variante multicritère sous contrainte de matroïde

Nous considérons le problème de couverture maximale prenant en entrée :

— Un ensemble d'éléments, noté $V = \{v_1, \ldots, v_q\}$ de taille q, où chaque élément v_i avec $i \in \{1, \ldots, q\}$ est évalué selon p critères.

— La collection des sous-ensembles (ou famille de sous-ensembles), notée $E = \{S_1, \ldots, S_n\}$ avec $n = |E|$ et où $S_i \subseteq V$ pour $i \in \{1, \ldots, n\}$.

L'utilité d'un élément $v \in V$, notée $u : V \to \mathbb{R}^+$, est définie par une somme pondérée de la façon suivante :

$$u^\lambda(v) = \sum_{i=1}^{p} \lambda_i u_i(v)$$

où $u_i(v)$ est l'évaluation de v sur le critère $i \in \{1, \ldots, p\}$ et $\lambda = (\lambda_1, \ldots, \lambda_p)$ un vecteur de poids normalisé.

L'objectif est alors de sélectionner des sous-ensembles de E afin que le poids total (ou la somme des utilités) des éléments couverts soit maximal. Une solution X est donc composée des éléments de E, soit $X \subseteq E$, et la fonction d'ensemble monotone sous-modulaire w à maximiser est définie pour toute solution X par :

$$w(X) = \sum_{v \in \bigcup\limits_{S \in X} S} u^\lambda(v)$$

Le MCP sous contrainte de matroïde uniforme restreint la cardinalité maximale d'une solution. C'est-à-dire que la collection des sous-ensembles indépendants $\mathcal{I}$ est défini tel que :

$$\mathcal{I} = \{X \subseteq E : |X| \leq k\}$$

avec $k > 0$ un entier donné.

Le MCP sous contrainte de matroïde de partition restreint la cardinalité maximale de la sélection dans chacune des m partitions disjointes de l'ensemble E. On note $\mathcal{D}_1, \ldots, \mathcal{D}_m$ les m partitions et $k_l > 0$ un entier donné pour tout $l = 1, \ldots, m$ la taille de la sélection dans chaque partition. Ainsi, nous avons $\sum_l^m = k_l = k$ et collection des sous-ensembles indépendants $\mathcal{I}$ est défini tel que :

$$\mathcal{I} = \{X \subseteq E : |X \cap \mathcal{D}_l| \leq k_l, \forall l \in \{1, \ldots, m\}\}$$

Dans la suite de cette section, on illustre le problème de couverture maximale sur différentes petites instances.

Illustration du MCP multicritère sous contrainte de matroïde uniforme

Exemple 4.3.1. *On considère une instance du MCP sous contrainte de matroïde uniforme constituée de $q = 10$ éléments, soit $V = \{v_1, \ldots, v_{10}\}$ et d'une collection de $n = 5$ sous-ensembles noté $E = \{S_1, \ldots, S_5\}$. La Table 4.1 présente les éléments couverts par chacun des sous-ensembles de E.*

S_1	S_2	S_3	S_4	S_5
v_2	v_1	v_4	v_4	v_7
v_3	v_2	v_5	v_8	v_9
v_{10}	v_6	v_8		
	v_{10}			

TABLE 4.1 – Famille E de sous-ensembles.

Ces ensembles sont également représentés graphiquement en Figure 4.1. Sous contrainte de matroïde uniforme, une solution réalisable $X \subseteq E$ est constituée de k sous-ensembles de E, c'est-à-dire $|X| \leq k$. Dans notre exemple on considère $k = 2$.

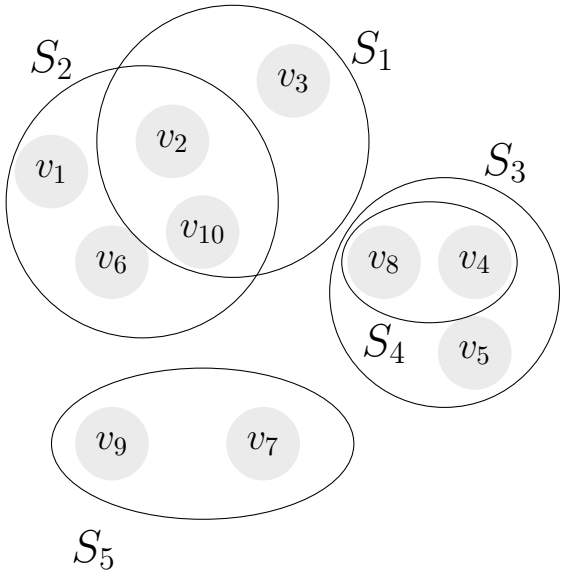


FIGURE 4.1 – Illustration d'une instance du MCP pour $q = 10$ et $n = 5$.

Si on considère le cas simple où l'utilité de chaque élément est égale à 1, $u^{\lambda}(v) = 1, \forall v \in V$, le problème revient à maximiser le nombre d'éléments couverts par les sous-ensembles sélectionnés.

On voit clairement ici que la solution optimale est $X = \{S_2, S_3\}$ *dont l'évaluation est égale à* 7. *En effet, si on sélectionne le sous-ensemble* S_2 *car il couvre* 4 *éléments, alors le sous-ensemble* S_1 *ne rapporte plus que* 1 *puisque les éléments* v_2 *et* v_{10} *sont déjà couverts par* S_2 *(précédemment sélectionné). Par conséquent, on sélectionne ensuite le sous-ensemble* S_3 *qui couvre* 3 *éléments.*

Illustrations des garanties de performances avec le MCP

Dans l'exemple précédent, on a vu que l'algorithme glouton classique trouve la solution exacte (on a choisit S_2 puis S_3). Malheureusement, cela n'est pas toujours le cas, comme vu théoriquement en Section 4.2.1 (garantie de performance de $1 - \frac{1}{e}$). Pour illustrer cela, considérons l'instance suivante du MCP sous contrainte de matroïde uniforme.

Exemple 4.3.2. *On considère l'instance suivante [Roughgarden, 2017] :*

- $V = \{v_1, v_2, v_3, v_4\}$
- $p = 1, u(v_i) = 1, \forall i \in \{1, \ldots, 4\}$
- $E = \{S_1, S_2, S_3\}$
- $S_1 = \{v_1, v_3\}$, $S_2 = \{v_2, v_4\}$, $S_3 = \{v_3, v_4\}$
- $k = 2$

On sélectionne l'ensemble comprenant le plus d'éléments, ici on a le choix entre S_1, S_2 *et* S_3. *Si le glouton choisit* S_3 *en premier, on a ensuite le choix entre* S_1 *et* S_2 *(qui couvre un seul élément de plus,* v_1 *pour* S_1, v_2 *pour* S_2*). La solution trouvée par le glouton (nommée* X*) couvre donc 3 éléments. Or on voit facilement que la solution optimale (*X^**) est* $\{S_1, S_2\}$ *qui couvre 4 éléments. On a donc* $w(X) = \frac{3}{4}w(X^*)$, *ce qui correspond exactement à la borne atteinte par le glouton (la borne exacte est égale à* $(1 - (1 - \frac{1}{r(M)})^{r(M)})$ *ce qui donne donc* $1 - (\frac{1}{2})^{\frac{1}{2}} = \frac{3}{4}$ *avec* $r(M) = k = 2$, .

Nous considérons maintenant une autre petite instance du MCP sous contrainte de matroïde de partition qui illustre le fait que la recherche locale ne donne pas toujours la solution optimale mais avec garantie de performance égale à $\frac{1}{2}$.

Exemple 4.3.3. *On considère l'instance suivante [Filmus and Ward, 2012] :*

- $V = \{v_1, v_2, v_3, v_4\}$
- $p = 1, u(v_1) = 1, u(v_2) = 1, u(v_3) = \varepsilon, u(v_4) = \varepsilon$ *avec* ε *une valeur positive proche de zéro*
- $E = \{S_1, S_2, S_3, S_4\}$
- $S_1 = \{v_1, v_3\}$, $S_2 = \{v_2\}$, $S_3 = \{v_4\}$, $S_4 = \{v_1\}$
- $m = 2 : \mathcal{D}_1 = \{S_1, S_2\}, \mathcal{D}_2 = \{S_3, S_4\}$
- $k_1 = 1, k_2 = 1$

Pour cette instance, on voit que la solution $X = \{S_1, S_3\}$ *d'évaluation égale à* $1+2\varepsilon$ *est un optimum local pour les échanges 1-1 puisque remplacer* S_1 *par* S_2 *donne la solution* $\{S_2, S_3\}$ *d'évaluation égale à* $1 + \varepsilon$ *et remplacer* S_3 *par* S_4 *donne la solution* $\{S_1, S_4\}$ *d'évaluation également égale à* $1 + \varepsilon$ *(perte d'une valeur* ε *dans les deux cas). Or on voit clairement que l'optimum global est la solution* $X^* = \{S_2, S_4\}$ *d'évaluation égale à 2. On a donc* $w(X) =$

$\frac{1+2\varepsilon}{2}w(X^*)$, *ce qui correspond à la borne atteinte par la recherche locale (borne égale à* $\frac{1}{2}$ *que l'on atteint en faisant tendre* $\varepsilon \to 0$*).*

Notons aussi que la solution $\{S_1, S_3\}$ *est aussi celle atteinte par l'algorithme glouton, on ne peut donc pas espérer améliorer la garantie de performance de la recherche locale en partant de la solution générée par l'algorithme glouton.*

4.3.2 Illustrations des algorithmes

Dans un premier temps, nous présentons l'application de notre algorithme glouton (cf. Algorithme 9) et de la recherche locale interactive (cf. Algorithme 8) sur une petite instance du problème de couverture maximale sous contrainte de matroïde uniforme.

Exemple 4.3.4. *Considérons une instance du problème de couverture maximale sur un matroïde uniforme avec un ensemble* $V = \{v_1, \ldots, v_q\}$ *composé de* $q = 10$ *éléments, et une famille de* $n = 8$ *sous-ensembles* $E = \{S_1, \ldots, S_n\}$ *définie par :*

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
v_3	v_1	v_6	v_2	v_7	v_6	v_2	v_1
v_4	v_3	v_{10}	v_8	v_9	v_7	v_8	v_3
v_5					v_{10}		v_5

TABLE 4.2 – Famille E de sous-ensembles.

Une solution réalisable est un ensemble de sous-ensembles $X \subseteq E$ *tel que* $|X| \leq k$ *(ici nous fixons* $k = 2$*), et le but est d'identifier une solution réalisable* X *maximisant* $w(X)$ *pour une fonction d'ensemble donnée* w *définie sur* 2^E *(selon l'équation page 132,* $w(X) = \sum_{v \in \bigcup_{S \in X} S} u^\lambda(v)$ *). Les valeurs* $u^\lambda(v)$ *sont définies à partir des utilités suivantes :*

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
u_1	4	2	2	3	7	6	8	7	7	1
u_2	5	7	1	2	3	1	5	1	9	1
u_3	4	5	3	7	2	5	3	8	4	4

TABLE 4.3 – Vecteur performance de chaque élément.

En fait, nous supposons que tous les éléments $v \in V$ *sont évalués par rapport à 3 critères (dénotés par* u_1*,* u_2*, et* u_3*), et que leurs évaluations sont données dans le Tableau 4.6.*

Ainsi l'utilité de tout élément $v \in V$ *est définie par :*

$$u^\lambda(v) = \sum_{i=1}^{3} \lambda_i u_i(v) \tag{4.5}$$

où $\lambda = (\lambda_1, \lambda_2, \lambda_3) \in \mathbb{R}_+$ représente le système de valeurs du décideur. On considère pour cet exemple que les préférences du décideur peuvent être représentées par la fonction d'ensemble w^* définie par le paramètre $\lambda^* = (0.2, 0.5, 0.3)$ qui est caché de la procédure. Nous commençons l'exécution sans connaissance sur les préférences du décideur, et nous devons donc considérer tous les jeux de poids possibles, soit tout jeu de poids λ dans l'ensemble $\Lambda_\Theta = \{\lambda \in [0,1]^3 : \sum_{i=1}^3 \lambda_i = 1\}$ et Θ les préférences collectées auprès du décideur (initialement vide), ce qui définit implicitement l'ensemble W. Nous considérons également que le seuil de tolérance est égal à 0, $\delta = 0$. On présente maintenant une exécution de l'algorithme glouton interactif.

Première itération : *On a $X = \emptyset$ et $E_c = E$, et donc $\mathcal{S} = E$. Puisque $MMR(\mathcal{S}, W) = 6 > 0$, on demande au décideur de comparer deux éléments de $\mathcal{S}$, soit S_5 et S_7 selon la stratégie CSS basée sur le minimax regret. Puisque nous avons $w^*(\{S_5\}) = 12.1 \geq 9.7 = w^*(\{S_7\})$, la réponse est : "le sous-ensemble S_5 est meilleur que le sous-ensemble S_7". Alors W est mis à jour en imposant la contrainte $w(\{S_5\}) \geq w(\{S_7\})$ qui revient à restreindre Λ_Θ (initialement représenté par le triangle Figure 4.2(a)) en imposant $\lambda_2 \geq \lambda_3 - \lambda_1$. Maintenant, Λ_Θ est représenté par le polyèdre BCDE de la Figure 4.2(b). Puisque $MMR(\mathcal{S}, W) = 2.5 > 0$, on demande au décideur de comparer deux sous-ensembles, disons S_5 et S_6. Puisque nous avons $w^*(\{S_5\}) = 12.1 \geq w^*(\{S_6\}) = 10.1$, le décideur répond : "Le sous-ensemble S_5 est meilleur que le sous-ensemble S_6". Ensuite, W est mis à jour en imposant la contrainte $w(\{S_5\}) \geq w(\{S_6\})$, ce qui revient à restreindre davantage Λ_Θ en imposant $\lambda_2 \geq \frac{5}{7}\lambda_3$. Maintenant, Λ_Θ est représenté par le polyèdre BCFE de la Figure 4.2(c). Nous avons $MMR(\mathcal{S}, W) = MR(\{S_5\}, \mathcal{S}, W) = 0$, et donc S_5 est ajouté à X.*

Seconde itération : *On a $X = \{S_5\}$ et $E_c = E \backslash \{S_5\}$, et donc $\mathcal{S} = \{\{S_5\} \cup \{S\} : S \in E_c\}$. Puisque $MMR(\mathcal{S}, W) = 1.5$, nous demandons au décideur de comparer deux éléments de $\mathcal{S}$, disons $\{S_5, S_8\}$ et $\{S_5, S_7\}$. Le décideur préfère la première option car $w^*(\{S_5, S_8\}) = 21.9 \geq w(\{S_5, S_7\}) = 21.8$. L'ensemble d'incertitude W est donc mis à jour en imposant $w(\{S_5, S_8\}) \geq w(\{S_5, S_7\})$, c'est-à-dire $\lambda_2 \geq \frac{2}{5}\lambda_3 - \frac{2}{5}\lambda_1$. Maintenant, Λ_Θ est représenté par le triangle BCG dans la Figure 4.2(d). Puisque nous avons $MMR(\mathcal{S}, W) = MR(\{S_5, S_8\}, \mathcal{S}, W) = 0$, alors le sous-ensemble S_8 est ajouté à X.*

Puisque $|X| = k = 2$, l'algorithme s'arrête et retourne $X = \{S_5, S_8\}$, qui est la solution optimale pour cette instance. Ceci indique que nous sommes capables de faire de bonnes recommandations en pratique sans connaître λ^* précisément (ici seulement 3 questions sont nécessaires).

Nous présentons maintenant une exécution de l'Algorithme 8 (recherche locale interactive) engendrant la même évolution dans l'espace Λ_Θ des jeux de poids admissibles.

Exemple 4.3.5. *Nous considérons la même instance du problème de couverture maximale décrite dans l'exemple précédent avec le seuil δ également fixé à 0. Supposons que* `ComputeInitialBasis` *renvoie $X = \{S_1, S_3\}$ dont l'évaluation multicritère est égale à $(19, 8, 21)$.*

Première itération : *la solution X compte 12 voisins qui sont obtenus en remplaçant S_1 par tout S_i avec $i \in \{2, 4, 5, 6, 7, 8\}$ (ce qui donne un nouvel ensemble de solutions appelé*

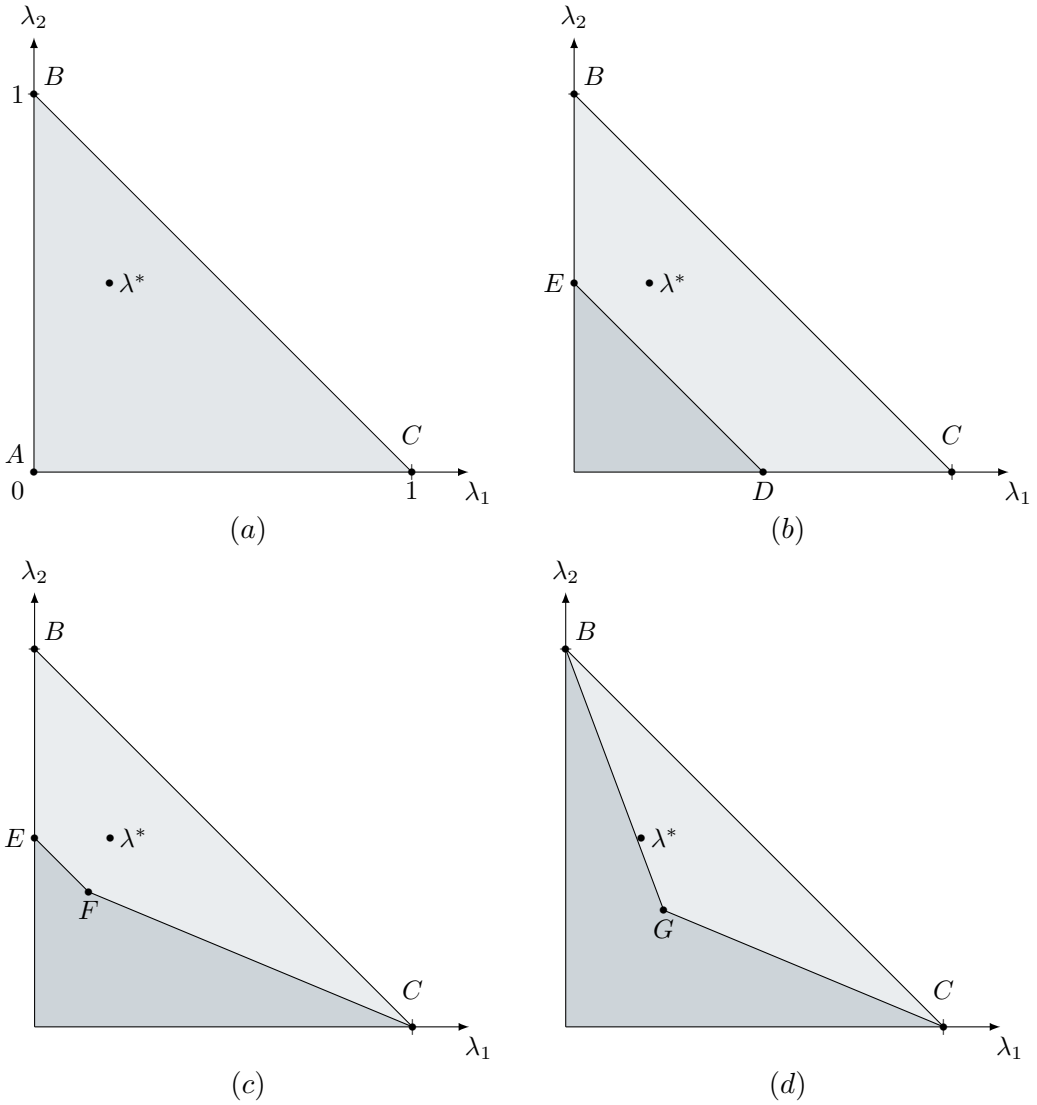


FIGURE 4.2 – Évolution de Λ_Θ au cours de l'exécution de l'algorithme 9 et de l'algorithme 8.

N_1) ou en remplaçant S_3 par par tout S_i avec $i \in \{2,4,5,6,7,8\}$ (ce qui donne un ensemble de solutions appelé N_3). On a donc $N_X = N_1 \cup N_3$. Puisque $MMR(N_X \cup \{X\}, W) = 6 > 0$, on demande au décideur de comparer deux éléments de $N_X \cup \{X\}$, disons $\{S_1, S_4\}$ à $\{S_1, S_5\}$. Nous avons $w^*(\{S_1, S_5\}) = 21.1 \geq w^*(\{S_1, S_4\}) = 18.7$, et donc nous devons mettre à jour Λ_Θ en imposant $w(\{S_1, S_5\}) \geq w(\{S_1, S_4\})$, c'est-à-dire $\lambda_2 \geq \lambda_3 - \lambda_1$ (voir la Figure 4.2(b) où les paramètres admissibles sont représentés par le polyèdre BCDE). Nous avons maintenant $MMR(N_X \cup \{X\}, W) = 2.5 \geq 0$, et nous pouvons donc demander au décideur de comparer $\{S_1, S_5\}$ et $\{S_1, S_6\}$. Puisque $w^*(\{S_1, S_5\}) = 21.1 \geq w^*(\{S_1, S_6\}) = 19.1$, nous ajoutons la contrainte $w(\{S_1, S_5\}) \geq w(\{S_1, S_6\})$, c'est-à-dire $\lambda_2 \geq \frac{5}{7}\lambda_3$ (voir la Figure 4.2(c)). Maintenant $MMR(N_X \cup \{X\}, W) = MR(\{S_1, S_5\}, N_X \cup \{X\}, W) = 0$, et

donc $X = \{S_1, S_5\}$ pour la prochaine itération.

Deuxième itération : *On a $N_X = N_1 \cup N_5$ où $N_j = \{\{S_j\} \cup \{S_i\} : i \in \{2,3,4,6,7,8\}\}$ pour $j \in \{1,5\}$. Puisque $MMR(N_X \cup \{X\}, W) = 1.5 > 0$, on demande au décideur de comparer deux éléments de $N_X \cup \{X\}$, disons $\{S_4, S_5\}$ et $\{S_5, S_8\}$. Nous avons $w^*(\{S_5, S_8\}) = 21.9 \geq w^*(\{S_4, S_5\}) = 21.8$, donc Λ_Θ est restreint par la contrainte $w(\{S_5, S_8\}) \geq w(\{S_4, S_5\})$, c'est-à-dire $\lambda_2 \geq \frac{2}{5}\lambda_3 - \frac{2}{5}\lambda_1$. L'ensemble des paramètres admissibles est représenté en Figure 4.2(d). Maintenant $MMR(N_X \cup \{X\}, W) = MR(\{S_5, S_8\}, N_X \cup \{X\}, W) = 0$, et donc $X = \{S_5, S_8\}$ pour la prochaine itération.*

Troisième itération : *On a $N_X = N_5 \cup N_8$ où $N_j = \{\{S_j\} \cup \{S_i\} : i \in \{1,2,3,4,6,7\}\}$ pour $j \in \{5,8\}$, et $MMR(N_X \cup \{X\}, W) = MR(X, N_X \cup \{X\}, W) = 0$. Par conséquent, il n'est pas nécessaire de poser de questions à cette étape, et l'algorithme se termine en retournant $X = \{S_5, S_8\}$, qui est la solution optimale pour cette instance.*

4.3.3 Résolution exacte avec paramètres connus

Afin d'évaluer nos méthodes de résolution interactive, nous allons déterminer les solutions optimales à jeu de poids connu à l'aide d'un programme linéaire en nombre entier (PLNE). Dans ce cas, le jeu de poids λ est connu du programme et représente le jeu de poids du décideur (caché dans nos procédures interactives).

Résolution du MCP sous contrainte de matroïde uniforme

Le PLNE pour le MCP pondéré sous contrainte de matroïde uniforme peut s'écrire de la façon suivante :

$$\begin{aligned}
\max \quad & \sum_{v_j \in V} w(v_j) y_j & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq k, & & (4.6)\\
& \sum_{S_i \subseteq E : v_j \in S_i} x_i \geq y_j, & \forall j \in \{1, \ldots, q\} & (4.7)\\
& y_j \in \{0,1\} & \forall j \in \{1, \ldots, q\} & (4.8)\\
& x_i \in \{0,1\} & \forall i \in \{1, \ldots, n\} & (4.9)
\end{aligned}$$

Le vecteur binaire $y = (y_1, \ldots, y_q)$ représente les éléments de V qui sont couverts ou non (si la variable y_j vaut 1, soit $y_j = 1$, alors l'élément v_j est couvert). Si la variable x_i vaut 1, soit $x_i = 1$, alors c'est le sous-ensemble $S_i \in E$ qui est sélectionné. La contrainte 4.6 garantit la sélection de maximum k sous-ensembles de la famille E dans notre solution finale (matroïde uniforme). La contrainte 4.7 permet d'imposer la sélection d'au moins un sous-ensemble S_i contenant un élément v_j couvert (en effet, si v_j est couvert alors $y_j = 1$ et au moins un sous-ensemble S_i contenant v_j est sélectionné). Enfin, les contraintes 4.8 et 4.9 indiquent que les x_i

et y_j sont des variables binaires. La fonction objectif permet de maximiser le poids total des éléments couverts, avec $w(v_j)$ qui représente ici le poids de l'élément v_j. On a ensuite choisi d'utiliser une autre variante du MCP, soit le MCP sous contrainte du matroïde de partition.

Résolution du MCP sous contrainte du matroïde de partition

Pour le matroïde de partition, la collection de sous-ensembles E est découpée en m ensembles disjoints $\mathcal{D} = \mathcal{D}_1, \ldots, \mathcal{D}_m$, et pour chaque sous-ensemble $\mathcal{D}_l$ au plus k_l sous-ensembles peuvent être sélectionnés. La définition du matroïde de partition est donnée en Section 3.2.1. Le PLNE de cette version du MCP peut s'écrire de la façon suivante :

$$\begin{aligned}
\max \quad & \sum_{v_j \in V} w(v_j) y_j & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} d_{il} x_i \leq k_l, & \forall l \in \{1, \ldots, m\} & \quad (4.10)\\
& \sum_{S_i \subseteq E : v_j \in S_i} x_i \geq y_j, & \forall j \in \{1, \ldots, q\} & \quad (4.11)\\
& y_j \in \{0, 1\} & \forall j \in \{1, \ldots, q\} & \quad (4.12)\\
& x_i \in \{0, 1\} & \forall i \in \{1, \ldots, n\} & \quad (4.13)
\end{aligned}$$

Seule la contrainte 4.10 diffère par rapport au PLNE du MCP sous contrainte de matroïde uniforme. On considère ici une nouvelle constante, une matrice binaire d_{il}, avec $d_{il} = 1$ si et seulement si l'ensemble S_i est contenu dans la partition $\mathcal{D}_l$. Ainsi, on impose que dans chaque partie disjointe $\mathcal{D}_l$ de E une solution ne compte pas plus de k_l sous-ensembles.

4.3.4 Résultats numériques

La suite de cette section est consacrée à la présentation des résultats obtenus par nos algorithmes. Nous considérons ici des instances du problème de couverture maximale pondéré avec un ensemble $V = \{v_1, \ldots, v_q\}$ de $q = 100$ éléments, et une famille E de $n = 80$ sous-ensembles de V. La génération de la famille de sous-ensembles est basée sur la méthode développée dans l'article de Resende [Resende, 1998] où la famille de sous-ensembles correspond au point d'installation d'éléments (des antennes par exemple) pouvant couvrir différents points de demande.

Plus précisément, avec cette méthode de génération d'instance, la dimension du problème est déterminée à l'aide de deux paramètres :

— f : le nombre de points de demande.

— e : le nombre d'emplacements potentiels d'installations.

Ces emplacements d'installations sont limités aux emplacements des points de demande, ainsi on a nécessairement $f \geq e$. En effet, les emplacements d'installations sont sélectionnés parmi les points de demande. Trois autres paramètres sont nécessaires à ce générateur, soient :

— c_{min} : le plus petit poids accordé à un point de demande.

— c_{max} : le plus grand poids accordé à une demande.
— r_{max} : distance maximale entre un emplacement d'installation et tout point de demande couvert par l'emplacement d'installation.

Les poids des points de demande sont distribués uniformément dans l'intervalle $[c_{min}, c_{max}]$ et les points de demande sont situés uniformément dans un carré unitaire. Parmi les f points de demande, un nombre e de points sont sélectionnés au hasard afin d'être des emplacements d'installation potentiels. Un emplacement potentiel d'installation possède des coordonnées cartésiennes (x_e, y_e) et couvre un point de demande situé aux coordonnées (x_v, y_v) s'ils sont à une distance (euclidienne) d'au plus r_{max} l'un de l'autre, c'est-à-dire si $\sqrt{(x_e - x_v)^2 + (y_e - y_v)^2} \leq r_{max}$.

Les expérimentations ont été réalisées sur des problèmes avec $f = 80$ et $e = 100$ (par analogie avec notre définition, f représente le nombre d'éléments soit la taille de l'ensemble V et e représente la taille de E). Les poids des points de demande ont été générés de manière aléatoire et uniforme dans l'intervalle [1, 10]. Les coordonnées des points de demande sont générés de manière uniforme entre 0 et 1. Le rayon r_{max} est fixé à 0.2 et trois valeurs différentes de p (le nombre de critères) sont considérées : $p = 4, 6$ et 8.

Les préférences du décideur sont alors représentées par une fonction d'ensemble w monotone sous-modulaire comme définie précédemment. Nous supposons ici que le jeu de poids λ est initialement inconnu. Les réponses aux questions sont simulées en utilisant un jeu de poids caché généré aléatoirement avant l'exécution des algorithmes. Pour un jeu de poids donné, la solution optimale est calculée, à l'aide des PLNE définis dans la section précédente.

Pour le matroïde uniforme, nous nous concentrons sur les sous-ensembles de taille maximale $k = 16$, c'est-à-dire $\mathcal{I} = \{X \subset E : |X| \leq 16\}$. Pour le matroïde de partition, l'ensemble E est partitionné aléatoirement en $m = 4$ ensembles $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$, et au plus $k_l = 4$ éléments peuvent être sélectionnés dans chaque ensemble $\mathcal{D}_l$, c'est-à-dire $\mathcal{I} = \{X \subseteq E : \forall l \in \{1, \ldots, 4\}, |X \cap \mathcal{D}_l| \leq 4\}$. Les résultats sont donnés dans le Tableau 4.4 et le Tableau 4.5 respectivement.

Pour notre algorithme de recherche locale, nous considérons deux implémentations de la procédure `ComputeInitialBasis` : génération aléatoire d'une base et génération d'une base à l'aide de l'algorithme glouton classique avec le jeu de poids uniforme $\lambda = (\frac{1}{p}, \ldots, \frac{1}{p})$.

Dans le Tableau 4.4, nous observons, pour $\delta = 0$, que l'algorithme glouton interactif est moins performant que l'algorithme de recherche locale. En effet, la méthode gloutonne est environ 10 fois plus lente (en moyenne) et pose plus de questions. De plus, nous observons que la recherche locale est plus performante lorsque l'on considère l'heuristique gloutonne plutôt que l'heuristique aléatoire pour générer la solution de départ. Dans ce tableau, nous avons également ajouté le nombre d'itérations réalisés par la recherche locale (voir la colonne notée NbI). Nous remarquons que le nombre d'itérations est logiquement beaucoup plus faible lorsque nous partons d'une solution de bonne qualité (lancement glouton) que d'une solution générée aléatoirement. Nous observons également que l'utilisation de $\delta = 20\%$ permet de réduire significativement le nombre de questions, sans vraiment augmenter le pourcentage d'erreur, excepté pour la recherche locale avec un point de départ aléatoire. Enfin, nous

		Glouton			Recherche locale (lancement aléatoire)				Recherche locale (lancement glouton)			
δ	p	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)	NbI	temps(s)	#questions	erreur (%)	NbI
0	4	4.4	19.4	1.1	0.2	17.2	0.9	12.6	0.2	12.1	0.5	2.9
	6	6.9	36.2	1.3	0.8	32.4	1.0	12.6	0.3	17.1	0.4	2.5
	8	9.9	48.3	1.1	1.4	42.9	0.8	12.5	0.5	28.5	0.3	3.0
0.2	4	2.8	7.2	1.3	0.1	7.6	6.1	7.7	0.1	9.5	0.5	2.9
	6	3.9	13.7	1.5	0.3	13.1	4.6	8.9	0.3	13.7	0.4	2.4
	8	5.0	17.3	1.3	0.6	19.5	4.0	9.4	0.4	21.8	0.3	2.9

TABLE 4.4 – Résultats obtenus pour le problème de couverture maximale sous matroïde uniforme avec l'algorithme glouton et la recherche locale interactifs (solutions initiales générées aléatoirement ou avec l'algorithme glouton) .

		Glouton			Recherche locale (lancement aléatoire)				Recherche locale (lancement glouton)			
δ	p	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)	NbI	temps(s)	#questions	erreur (%)	NbI
0	4	3.8	20.5	4.2	0.2	16.2	4.1	11.1	0.1	9.0	2.2	2.8
	6	5.4	33.4	3.5	0.6	25.5	4.5	10.7	0.2	13.4	1.8	2.7
	8	6.8	44.3	4.1	0.9	35.9	4.3	10.6	0.3	17.7	2.2	3.0
0.2	4	2.4	7.4	4.3	0.1	7.5	6.8	7.7	0.1	7.2	2.3	2.7
	6	3.1	12.4	3.9	0.2	10.6	7.2	7.6	0.2	10.3	1.9	2.7
	8	4.1	17.8	4.5	0.4	15.9	5.9	8.8	0.3	14.6	2.2	3.0

TABLE 4.5 – Résultats obtenus pour le problème de couverture maximale sous matroïde de partition avec l'algorithme glouton et la recherche locale interactifs (solutions initiales générées aléatoirement ou avec l'algorithme glouton).

observons que nos algorithmes sont plus performants sur le matroïde uniforme que sur le matroïde de partition qui est un peu plus complexe (cf. Tableau 4.5). Pour le matroïde de partition, nous avons les mêmes conclusions que pour le matroïde uniforme (nous observons juste des temps d'exécution et des pourcentages d'erreur légèrement plus élevés).

4.4 Application au problème de sélection collective

4.4.1 Présentation et formalisation du problème étudié

Dans le problème de sélection collective de sous-ensembles, on dispose d'un ensemble A de m agents, et d'un ensemble $E = \{e_1, \ldots, e_n\}$ de n éléments. Chaque agent $a \in A$ donne un score $s_a(e) \geq 0$ à chaque élément $e \in E$, et l'utilité que l'agent a tire d'un ensemble $X \subseteq E$ est définie par une moyenne pondérée ordonnée (OWA) [Yager, 1988]. Plus précisément, pour tout $X = \{x_1, \ldots, x_k\} \subseteq E$ de taille $k \leq n$, l'utilité de l'agent a est définie par :

$$u_a^{\lambda}(X) = \sum_{i=1}^{k} \lambda_i s_a(x_{(i)})$$

où $(\cdot)$ est une permutation de $\{1, \ldots, k\}$ triant les éléments de X par score non croissant (c'est-à-dire, $s_a(x_{(1)}) \geq \ldots \geq s_a(x_{(k)})$), et $\lambda = (\lambda_1, \ldots, \lambda_n) \in [0,1]^n$ est un vecteur normalisé

non croissant. Ici, la fonction d'ensemble w est simplement définie comme la somme des utilités de chaque agent :

$$w(X) = \sum_{a \in A} u_a^{\lambda}(X)$$

Proposition 4.4.1. *Soit une instance du problème de sélection collective avec un OWA à vecteur de poids λ non croissant. La fonction w est sous-modulaire (voir [Skowron et al., 2016] pour une preuve).*

On illustre ce problème dans l'exemple suivant.

Exemple 4.4.1. *On considère une petite instance du problème de sélection collective avec un ensemble de 5 agents ($m = 5$) noté $A = \{a_1, a_2, a_3, a_4, a_5\}$ et un ensemble $E = \{e_1, e_2, e_3, e_4\}$ constitué de $n = 4$ éléments. Les scores des éléments pour chaque agent sont donnés dans le Tableau 4.6.*

	$s_a(e_1)$	$s_a(e_2)$	$s_a(e_3)$	$s_a(e_4)$
a_1	4	2	3	1
a_2	4	2	3	1
a_3	1	3	2	4
a_4	2	4	1	3
a_5	2	4	1	3

TABLE 4.6 – Utilités des agents pour les différents éléments.

On considère $k = 3$ et on utilise comme jeu de poids pour le décideur un OWA avec poids décroissant soit $\lambda = (0.5, 0.3, 0.2)$. Les agents 1 et 2 ainsi que les agents 4 et 5 ont les mêmes utilités. Il y a 4 solutions possibles, soient les ensembles suivants : $\{e_1, e_2, e_3\}$, $\{e_1, e_2, e_4\}$, $\{e_1, e_3, e_4\}$ et $\{e_2, e_3, e_4\}$.

Ainsi, calculer le score de la solution se décompose en trois calculs :
- $4 \times 0.5 + 2 \times 0.3 + 1 \times 0.2 = 2.8$ *pour les agents 1 et 2.*
- $4 \times 0.5 + 3 \times 0.3 + 1 \times 0.2 = 3.1$ *pour l'agent 3.*
- $4 \times 0.5 + 3 \times 0.3 + 2 \times 0.2 = 3.3$ *pour les agents 4 et 5.*

On peut conclure que la valeur de la solution X est $w(X) = 15.3$ ($2 \times 2.8 + 3.1 + 2 \times 3.3$).

C'est la solution optimale pour ce petit problème puisque les valeurs des trois autres solutions possibles, soient $\{e_1, e_2, e_3\}$, $\{e_1, e_3, e_4\}$ et $\{e_2, e_3, e_4\}$, sont respectivement égales à 14.5, 13.6 et 14.5.

4.4.2 Résolution exacte avec paramètres connus

De la même façon que pour le MCP, dans le but de connaître la distance à l'optimale de nos méthodes, nous devons avoir une méthode de résolution exacte (avec les paramètres du décideur connus). Soit n le nombre d'objets, g le nombre d'agents et k la taille de l'ensemble

cherché. Alors la formulation par programmation linéaire en nombres entiers de ce problème peut s'écrire de la manière suivante :

$$\max \quad \sum_{i=1}^{n}\sum_{j=1}^{g}\sum_{t=1}^{k} w_{ij}x_{ijt}$$

$$\text{s.t.} \quad \sum_{j=1}^{g} x_j = k, \qquad \forall j \in \{1,\dots,g\} \tag{4.14}$$

$$x_{ijk} \leq x_j, \qquad \forall i \in \{1,\dots,n\}; \forall j \in \{1,\dots,g\}; \forall k \in \{1,\dots,t\} \tag{4.15}$$

$$\sum_{j=1}^{g} x_{ijt} = 1, \qquad \forall i \in \{1,\dots,n\}; \forall t \in \{1,\dots,k\} \tag{4.16}$$

$$\sum_{t=1}^{k} x_{ijt} = 1, \qquad \forall i \in \{1,\dots,n\}; \forall j \in \{1,\dots,g\} \tag{4.17}$$

$$x_{ijt} \in \{0,1\}, \qquad \forall i \in \{1,\dots,n\}; \forall j \in \{1,\dots,g\}; \forall t \in \{1,\dots,k\} \tag{4.18}$$

$$x_j \in \{0,1\}, \qquad \forall j \in \{1,\dots,g\} \tag{4.19}$$

La variable x_{ijt} définie en contrainte (4.19) indique, si elle vaut 1, que pour l'agent i, l'élément e_j est le $t^{\text{ème}}$ meilleur parmi ceux qui sont sélectionnés dans la solution. La variable x_j quant à elle indique en valant 1 que l'élément e_j est inclus dans la solution. La contrainte (4.15) permet la cohérence des deux variables et la contrainte (4.14) impose que le nombre d'éléments sélectionnés soit égal à k. La contrainte (4.16) annonce que chaque agent ne classe qu'un seul des éléments de la solution comme $t^{\text{ème}}$ meilleur. Enfin, la contrainte (4.17) impose qu'il n'y ait pas d'agent avec un élément classé à deux positions différentes parmi ceux de la solution.

Comme on cherche une base de cardinalité k, cela correspond à un problème sous contrainte de matroïde uniforme. Si l'on souhaite modéliser ce problème sous-contrainte de matroïde de partition, l'ensemble E est alors divisé en m sous-ensembles disjoints notés $\mathcal{D}_1,\dots,\mathcal{D}_m$ et tels que $\sum_{l=1}^{m} k_l = k$. Ainsi, la contrainte (4.14) doit être modifiée de la façon suivante :

$$\sum_{j=1}^{g} x_j d_{jl} \leq k_l, \qquad \forall l \in \{1,\dots,m\} \tag{4.20}$$

Pareillement au MCP, on considère une matrice binaire d_{jl}, avec $d_{jl} = 1$ si et seulement si l'élément e_j est contenu dans la partition $\mathcal{D}_l$. Ainsi, cette contrainte (4.20) permet en remplaçant (4.14), que seul le nombre maximal de sélection soit faite par partition.

4.4.3 Résultats expérimentaux

Nous considérons des instances avec $g = 50$ agents et $n = 50$, et les scores sont générés aléatoirement dans $[1, 100]$. Pour le matroïde uniforme, deux valeurs de k ont été testées :

$k = 5$, et $k = 10$. Pour le matroïde de partition, E est divisé aléatoirement en $m = 4$ ensembles, et au plus $\frac{d}{4}$ éléments peuvent être sélectionnés dans chaque ensemble ; nous considérons deux valeurs de d ($d = 8$ et $d = 16$).

		Glouton			Recherche locale (lancement aléatoire)				Recherche locale (lancement glouton)			
δ	p	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)	NbI	temps(s)	#questions	erreur (%)	NbI
0	5	0.9	8.2	0.2	3.1	14.4	0.1	5.8	1.6	11.4	0.0	3.1
	10	2.7	25.1	0.1	34.4	45.6	0.0	9.3	31.2	35.5	0.0	4.3
0.2	5	0.6	3.0	0.2	0.8	6.8	0.5	5.0	1.1	6.9	0.1	2.9
	10	1.2	4.3	0.1	12.8	16.6	0.6	7.5	17.7	18.5	0.0	4.1

TABLE 4.7 – Problème de sélection collective d'éléments sous contrainte de matroïde uniforme.

		Glouton			Recherche locale (lancement aléatoire)				Recherche locale (lancement glouton)			
δ	p	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)	NbI	temps(s)	#questions	erreur (%)	NbI
0	8	1.3	14.0	0.2	2.2	24.0	0.0	8.1	1.5	17.7	0.0	3.8
	16	3.1	38.9	0.1	21.5	56.9	0.0	12.0	0.4	34.0	0.0	4.1
0.2	8	0.8	4.2	0.2	1.8	18.5	0.1	8.1	1.2	13.4	0.1	3.7
	16	1.4	6.3	0.1	16.0	43.4	0.2	11.4	6.9	25.7	0.0	4.0

TABLE 4.8 – Problème de sélection collective d'éléments sous contrainte de matroïde de partition.

Les résultats sont donnés dans le Tableau 4.7 et le Tableau 4.8. Cette fois, nous observons que l'algorithme glouton est plus rapide que la recherche locale. La recherche locale partant d'une solution de départ donnée par la méthode gloutonne est beaucoup plus efficace que la recherche locale partant d'une solution aléatoire, en termes de temps, de nombre de questions et pourcentage d'erreur, avec un nombre d'itérations moins important (noté NbI dans les tableaux). Pour les deux matroïdes, les algorithmes obtiennent de petites erreurs (moins de 0.6%). Avec $\delta = 20\%$, l'algorithme glouton est très efficace : des erreurs inférieures à 0.2% sont obtenues en 6.3 questions maximum en moyenne.

4.5 Cas de la minimisation d'une fonction sous-modulaire

Les problèmes que nous avons traité jusqu'à présent dans ce chapitre ne concernent que la maximisation d'une fonction sous-modulaire monotone sous contrainte de matroïde. Il est également possible d'étudier la *minimisation* de fonctions sous-modulaires (ce qui revient à maximiser une fonction super-modulaire, puisqu'une fonction $g(X)$ est super-modulaire si $-g(X)$ est sous-modulaire [Cunningham, 1985]). Étonnement, les propriétés des algorithmes pour le cas de la minimisation diffèrent fortement des algorithmes développés pour la maximisation [Iyer et al., 2013]. Par exemple, il existe des algorithmes polynomiaux pour résoudre le problème de la minimisation d'une fonction sous-modulaire monotone sans contrainte [McCormick, 2005]. Ainsi le problème de la recherche d'une coupe de coût minimal dans un graphe peut être résolu avec l'algorithme de Stoer–Wagner [Stoer and Wagner, 1997] en temps polynomial, tandis que la recherche d'une coupe de valeur maximale dans un graphe

est NP-difficile [Commander, 2009]. Par contre, sous contrainte, la minimisation de fonctions sous-modulaires devient plus difficile. Par exemple, l'application de l'algorithme glouton pour la minimisation d'une fonction sous-modulaire $w(X)$ sous contrainte de matroide uniforme donne la garantie suivante [Bai and Bilmes, 2018] :

$$w(X) \leq \frac{1}{1-\mathcal{K}_f} w(X^*) \text{ avec } \mathcal{K}_f = 1 - \min_{e \in E} \frac{\Delta(e|E \setminus \{e\})}{w(e)}$$

$\mathcal{K}_f$ est une mesure de la courbure de la fonction sous-modulaire, avec des valeurs comprises entre 0 et 1. Plus la courbure est proche de 0, plus la fonction est proche d'être modulaire (c'est-à-dire linéaire). Ainsi pour une fonction totalement linéaire ($\mathcal{K}_f = 0$) on a bien la garantie que l'algorithme glouton trouve toujours la solution optimale, alors que pour une fonction sous-modulaire très "courbée", avec une valeur de $\mathcal{K}_f$ proche de 1, il n'y a plus aucune garantie puisque le terme $\frac{1}{1-\mathcal{K}_f}$ tend vers l'infini lorsque $\mathcal{K}_f$ tend vers 1 (alors que l'on a une garantie de $1 - \frac{1}{e}$ quelle que soit la courbure de la courbe pour la maximisation).

Dans cette section, nous allons considérer les deux mêmes problèmes étudiés précédemment (problème de couverture et problème de sélection collective) mais en changeant simplement le sens de l'optimisation (minimisation au lieu de la la maximisation).

Nous considérons donc le problème de couverture vu en Section 4.3 mais en minimisation (problème de couverture minimale). Ainsi, au lieu de choisir k sous-ensembles pour maximiser la somme des poids des éléments couverts, nous cherchons à minimiser la somme des poids des éléments couverts. Ce problème, aussi connu sous le nom de *Minimum k-Union problem* [Chlamtác et al., 2017], semble naturel mais est assez peu étudié, sans doute à cause de sa difficulté par rapport au problème de couverture maximale. Il existe pourtant certaines applications liées à ce problème. Par exemple, les sous-ensembles S_i peuvent représenter des tâches à réaliser, où chaque tâche nécessite des ressources v_j (des machines par exemple). Si la mobilisation d'une machine a un coût fixe, il est nécessaire de minimiser le nombre de machines à utiliser, et donc de choisir des tâches possédant un maximum de machines en commun afin de minimiser les coûts.

Pour illustrer la difficulté de ce problème, considérons l'exemple suivant.

Exemple 4.5.1. *Soit l'instance suivante :*

- $V = \{v_1, v_2, \ldots, v_{16}\}$
- $p = 1, u(v_i) = 1, \forall i \in \{1, \ldots, 16\}$
- $E = \{S_1, S_2, \ldots, S_8\}$
- $S_1 = \{v_1, v_2, v_3\}$, $S_2 = \{v_4, v_5, v_6\}$, $S_3 = \{v_7, v_8, v_9\}$, $S_4 = \{v_{10}, v_{11}, v_{12}\}$, $S_5 = \{v_{13}, v_{14}, v_{15}\}$, $S_6 = \{v_{13}, v_{14}, v_{16}\}$, $S_7 = \{v_{13}, v_{15}, v_{16}\}$, $S_8 = \{v_{14}, v_{15}, v_{16}\}$
- $k = 4$ *(matroïde uniforme)*

Pour cette instance, chaque sous-ensemble S_i ($\forall i \in \{1, \ldots, 8\}$) couvre trois éléments. Si l'algorithme glouton sélectionne S_1, puis S_2, puis S_3 et enfin S_4 (ce qui augmente le coût de 3 à chaque fois) on obtient une solution X d'évaluation égale à 12. Or la solution optimale X^* est égale $\{S_5, S_6, S_7, S_8\}$ d'évaluation égale à 4. À cause de son comportement local, l'algorithme glouton ne voit donc pas qu'il existe des intersections non vides entre les sous-ensembles S_5, S_6, S_7 et S_8 et qu'il est intéressant de prendre plusieurs de ces sous-ensembles. L'algorithme

glouton peut donc générer une solution de très mauvaise qualité. Pour cette instance, puisque la valeur de $\min_{e \in E} \frac{\Delta(e|E\setminus\{e\})}{w(e)} = 0$ *(en prenant* $e = S_5$*,* S_6*,* S_7 *ou* S_8*) la valeur de* K_f *est égale à 1. Il n'y a donc aucune garantie de performance théorique pour l'algorithme glouton.*

Étant donné les mauvaises propriétés de l'algorithme glouton pour la minimisation de fonctions sous-modulaires, nous avons également étudié une nouvelle approche, basée sur les super-gradients de la fonction sous-modulaire.

4.5.1 Approche basée sur les super-gradients

En s'appuyant sur différentes inégalités développées par Nemhauser et al [Nemhauser et al., 1978] permettant de borner les fonctions sous-modulaires, Iyer *et al* ont développé une méthode de recherche locale basée sur les super-gradients pour optimiser une fonction sous-modulaire sous contrainte [Iyer et al., 2013]. L'approche est générique et peut être utilisée aussi bien en maximisation qu'en minimisation, avec ou sans contrainte. Nous allons ici particulièrement développer cette méthode dans le cas de la minimisation d'une fonction sous-modulaire sous contrainte. L'idée principale est d'optimiser une fonction linéaire bornant la fonction sous-modulaire.

L'algorithme est donné en Algorithme 10. Il consiste simplement à partir d'une solution réalisable du problème considéré (une base dans le cas de contraintes de type matroïde), et d'ensuite répéter les opérations suivantes :

1. Définition d'une fonction modulaire dépendant de la solution courante.
2. Recherche d'une solution optimale pour cette fonction modulaire (avec l'algorithme glouton par exemple qui est exact quand la fonction d'évaluation est linéaire). La solution trouvée devient la nouvelle solution courante.

La méthode s'arrête lorsque la solution optimale pour la fonction modulaire est la même solution que la solution courante.

Algorithme 10 Méthode de descente basée sur les super-gradients pour la minimisation de fonctions sous-modulaires

1: $X^0 \leftarrow$ solutionAleatoire()
2: **while** convergence pas atteinte (soit $X^t \neq X^{t-1}$) **do**
3: Définition d'une fonction modulaire m_{X^t} sur X^t
4: $X^{t+1} = \text{argmin}_{X \in 2^E}\ m_{X^t}(X)$
5: $t \leftarrow t + 1$
6: **end while**

La fonction modulaire est définie de la façon suivante. Soit X^t une solution réalisable. On a alors l'inégalité suivante, pour toute solution X réalisable [Iyer et al., 2013] :

$$f(X) \leq m_{X^t}(X) = f(X^t) + \sum_{j \in X} g^i_{X^t}(j) - \sum_{j \in X^t} g^i_{X^t}(j)$$

avec les quantités $g^i_{X^t}(j)$ ($i \in \{1,2,3\}$), représentant différents super-gradients, définis de la manière suivante :

$j \in X^t$	$j \notin X^t$
$g^1_{X^t}(j) = \Delta(j\|E \setminus \{j\})$	$g^1_{X^t}(j) = \Delta(j\|X^t)$
$g^2_{X^t}(j) = \Delta(j\|X^t \setminus \{j\})$	$g^2_{X^t}(j) = \Delta(j\|\emptyset)$
$g^3_{X^t}(j) = \Delta(j\|E \setminus \{j\})$	$g^3_{X^t}(j) = \Delta(j\|\emptyset)$

TABLE 4.9 – Super-gradients

Le super-gradient à utiliser (numéro 1, 2 ou 3) dépend du problème pour lequel la méthode est appliquée (les performances de la méthode peuvent varier en fonction du numéro de gradient). Notons aussi que l'on a $m_{X^t}(X^t) = f(X^t)$, donc la borne est serrée pour la solution courante.

Ci-dessous nous donnons un exemple d'application de cette méthode au problème de couverture minimale, en prenant le super-gradient numéro 2.

Exemple 4.5.2. *Nous reprenons l'exemple 4.3.2 avec les coûts suivants pour les différents éléments (pour cet exemple, les coûts sont ici connus et il n'y a donc pas d'élicitation) :*

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
u	1	3	10	2	9	6	10	1	5	2

Nous partons de la solution $X^0 = \{S_1, S_2\}$ dont l'évaluation est égale à $u(v_3) + u(v_4) + u(v_5) + u(v_1) = 22$.

Première itération : *Nous définissons la fonction modulaire $m_{X^0}(X) = f(X^0) + \sum_{j\in X} g^2_{X^0}(j) - \sum_{j\in X^0} g^2_{X^0}(j)$ que nous cherchons à minimiser. Comme $f(X^0)$ et $\sum_{j\in X^0} g^2_{X^0}(j)$ sont des constantes, il suffit de trouver une solution X qui minimise $\sum_{j\in X} g^2_{X^0}(j)$. On calcule donc les différentes quantités $g^2_{X^0}(j)$ pour $j \in E$. On obtient :*

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
$g^2_{X^0}(.)$	21	1	8	4	15	18	4	20

En sélectionnant les 2 sous-ensembles avec les coûts les plus faibles, on obtient la solution $\{S_2, S_4\}$, qui devient la prochaine solution courante (X^1). Cette solution a une évaluation égale à $w(X^1) = 15$.

Deuxième itération : *Nous définissons la fonction modulaire $m_{X^1}(X) = f(X^1) + \sum_{j\in X} g^2_{X^1}(j) - \sum_{j\in X^1} g^2_{X^1}(j)$ que nous cherchons à minimiser. On calcule donc les différentes quantités $g^2_{X^1}(j)$ pour $j \in E$. On obtient :*

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
$g^2_{X^1}(.)$	21	11	8	4	15	18	4	20

En sélectionnant les 2 sous-ensembles avec les coûts les plus faibles, on obtient la solution $\{S_4, S_7\}$*, qui devient la prochaine solution courante* (X^2)*. Cette solution a une évaluation égale à* $w(X^2) = 4$*.*

Troisième itération : *Nous définissons la fonction modulaire* $m_{X^2}(X) = f(X^2) + \sum_{j \in X} g^2_{X^2}(j) - \sum_{j \in X^2} g^2_{X^2}(j)$ *que nous cherchons à minimiser. On calcule les différentes quantités* $g^2_{X^2}(j)$ *pour* $j \in E$ *:*

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
$g^2_{X^2}(.)$	21	11	8	0	15	18	0	20

On obtient donc de nouveau la solution $\{S_4, S_7\}$*, qui est la solution optimale pour cette instance (que l'on peut facilement obtenir par énumération de tous les sous-ensembles de taille 2).*
La garantie de performance de cette méthode est la même que l'algorithme glouton, c'est-à-dire :

$$w(X) \leq \frac{1}{1 - \mathcal{K}_f} w(X^*)$$

4.5.2 Résultats numériques

Problème de couverture minimale pondéré

Les résultats de l'application de l'algorithme glouton, de la recherche locale (avec solution de départ générée aléatoirement) et de la méthode basée sur les super-gradients (avec le super-gradient numéro 2 qui donne les meilleurs résultats) sont donnés à la Figure 4.10 pour le problème de couverture minimal pondéré. Les instances utilisées sont les mêmes instances que celles décrites en Section 4.3.3. Uniquement le sens d'optimisation a été changé (maximisation $\rightarrow$ minimisation). On voit que par rapport au problème de maximisation, les erreurs sont beaucoup plus grandes, mais que les temps de calcul et nombre de questions restent similaires. Les erreurs obtenues par la méthode basée sur les super-gradients sont malheureusement très élevées (plus de 50% d'erreur), et cette méthode ne semble donc pas très prometteuse pour résoudre ce problème.

Problème de sélection collective

Les résultats pour le problème de sélection collective (en minimisation, les utilités sont remplacées par des coûts) sont donnés en Figure 4.11. De nouveau, les erreurs sont beaucoup plus importantes que pour la version en maximisation.

		Glouton			Recherche locale (lancement aléatoire)				Gradient 2			
δ	p	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)	NbI	temps(s)	#questions	erreur (%)	NbI
0	4	5.3	19.9	27.7	0.3	11.7	29.2	10.9	0.7	27.8	62.3	2.8
	6	7.9	32.6	28.2	0.5	14.6	31.2	10.8	1.6	46.5	60.4	2.9
	8	11.4	47.5	25.8	0.8	16.9	33.5	11.1	2.5	66.4	56.4	2.8
0.2	4	3.7	10.7	25.0	0.3	8.7	38.9	9.8	0.4	2.3	68.5	2.8
	6	4.8	14.4	26.2	0.5	10.5	38.5	9.9	0.7	2.9	61.1	3.1
	8	5.4	14.1	26.2	0.6	12.2	38.6	10.4	0.8	4.2	57.5	3.0

TABLE 4.10 – Résultats obtenus pour le problème de couverture minimal pondéré.

		Glouton			Recherche locale (lancement aléatoire)				Gradient 2			
δ	d	temps(s)	#questions	erreur (%)	temps(s)	#questions	erreur (%)	NbI	temps(s)	#questions	erreur (%)	NbI
0	5	1.2	6.4	5.4	3.3	6.4	4.8	5.7	0.1	3.7	5.7	2.4
	10	2.9	14.3	4.1	232.8	12.0	4.1	9.3	0.3	7.3	2.8	2.3
0.2	5	1.3	6.3	5.4	3.3	6.4	4.8	5.7	0.1	3.7	5.7	2.4
	10	3.2	13.7	4.2	210.4	12.6	3.9	9.4	0.3	5.7	2.8	2.3

TABLE 4.11 – Résultats obtenus pour le problème de sélection collective.

4.6 Conclusion du chapitre

Nous avons proposé deux algorithmes interactifs (glouton et recherche locale) combinant l'élicitation d'une fonction sous-modulaire et la détermination d'un sous-ensemble indépendant optimal sous contrainte de matroïde. Les deux algorithmes admettent des garanties de performances sur la qualité de la base retournée.

Le compromis entre la qualité des solutions et le nombre de questions de préférences utilisées dans le processus peut être contrôlé par le paramètre δ utilisé pour définir les regrets maximaux admissibles. Les deux algorithmes de décision interactifs proposés ici reposent sur la réduction progressive de l'ensemble des fonctions admissibles jusqu'à ce qu'une base optimale nécessaire (ou quasi-optimale) puisse être identifiée. À chaque étape, les réductions de l'espace des paramètres sont dues aux nouvelles déclarations de préférences obtenues du décideur. Cela rend le processus de décision sensible aux éventuelles erreurs dans les réponses. Une suite possible de ce travail serait de développer une approche plus flexible et tolérante aux erreurs dans la comparaison des solutions. Le défi serait de conserver la possibilité de fournir une bonne garantie sur la qualité de la solution finale tout en conservant de courtes séquences de questions au décideur dans le but d'acquérir des préférences dans le processus d'élicitation. Nous avons également vu que les algorithmes pour la minimisation d'une fonction sous-modulaire donnent une garantie de performance dépendant de la courbure de la fonction sous-modulaire. Sur les deux problèmes étudiés, ces algorithmes obtiennent des erreurs assez élevées. Il serait donc intéressant d'adapter les métaheuristiques développées au Chapitre 2 pour ces problèmes afin de réduire l'erreur en pratique.

Remarquons enfin que nous avons testé notre approche sur deux problèmes spécifiques, mais que de nombreux autres pourraient être résolus avec des performances similaires en raison de la généralité des matroïdes.

Conclusion et perspectives

Les travaux menés durant cette thèse proposent plusieurs méthodes d'élicitation interactives des préférences pour des problèmes d'optimisation combinatoire multi-objectifs. Les contributions se divisent en deux axes principaux : une première partie qui est basée sur des métaheuristiques et une seconde partie basée sur des méthodes à garanties de performances appliquées à des problèmes qui se modélisent sous la forme d'une structure de matroïde.

Le premier chapitre présente les domaines de l'optimisation combinatoire multi-objectifs et de l'aide à la décision, nécessaires à la compréhension de cette thèse. Ce chapitre met en exergue la variété d'approches pouvant être utilisées pour résoudre des problèmes multi-objectifs, et développe les principales approches de résolution exacte et par métaheuristique. Ce chapitre présente également les modèles décisionnels qui ont été étudiés lors de cette thèse.

Le deuxième chapitre décrit deux méthodes pour résoudre des problèmes combinatoires multi-objectifs avec préférences imprécisément connues. Le principe commun aux deux méthodes est d'intercaler des questions sur les préférences du décideur avec des étapes d'optimisation afin de concentrer l'effort d'élicitation sur l'obtention des informations préférentielles nécessaires à la détermination d'une bonne solution pour le décideur. Nous avons d'abord proposé une approche générale basée sur la recherche locale et l'élicitation incrémentale de préférences. L'avantage de cette méthode est qu'elle est applicable à des problèmes complexes pour lesquels il n'existe pas forcément de méthode efficace pour les résoudre avec préférences connues. À la suite de ces premiers résultats encourageants, nous avons proposé une autre méthode heuristique interactive, utilisant cette fois-ci l'approche génétique. Cette méthode nécessite de bénéficier d'un algorithme de résolution (exacte ou approché) du problème considéré. Ces deux méthodes ont été comparées à des algorithmes issus de l'état de l'art et ont données des résultats très satisfaisants sur des instances du voyageur de commerce et du sac à dos multi-objectifs avec une somme pondérée, un OWA et une intégrale de Choquet. De plus, notre algorithme génétique interactif a obtenu de meilleurs résultats que ceux observés avec la recherche locale et peut être utilisé sur de plus grandes instances.

Le troisième chapitre introduit deux méthodes d'optimisation interactives combinant l'élicitation des préférences avec l'exploration d'ensembles indépendants dans un matroïde, l'une basée sur la recherche locale et l'autre sur la recherche gloutonne. Ici, questions et optimisation s'entremêlent afin de déterminer une solution optimale ou une solution quasi-optimale si l'on veut économiser quelques questions. Lors de l'exécution de l'algorithme glouton, les

préférences collectées auprès du décideur permettent d'orienter la construction de la solution, pour la recherche locale cela permet d'orienter la recherche dans le voisinage. Les deux algorithmes permettent la détermination d'une base optimale mais admettent une version permettant d'économiser une quantité significative de questions si un pourcentage d'erreur est toléré. L'intérêt de notre proposition est qu'elle est assez générale et peut être mise en œuvre dans divers problèmes d'optimisation impliquant une structure matroïde. Nous avons implémenté ces méthodes sur trois problèmes (sélection de sous-ensembles, arbre couvrant et ordonnancement) sous différentes structures de matroïdes (uniforme, graphique et transversale). Enfin, nous présentons des tests numériques montrant l'efficacité pratique de nos algorithmes en termes de temps de calcul, de nombre de questions et d'erreur empirique sur ces problèmes.

Dans le quatrième chapitre, nous proposons une adaptation des deux méthodes interactives du chapitre précédent (une recherche gloutonne et une recherche locale) combinant l'élicitation d'une fonction d'utilité sous-modulaire et la détermination du sous-ensemble indépendant avec approximation dans un matroïde pondéré. Ici, à la différence du cas linéaire, c'est un approximation qui est retournée par nos algorithmes. Nous avons testé ces deux approches sur deux problèmes spécifiques, le problème de couverture maximum et le problème de sélection collective, mais que de nombreux autres pourraient être résolus avec des performances similaires en raison de la généralité des matroïdes. On note que nous avons également étudié la minimisation d'une fonction sous-modulaire sur ces mêmes problèmes à l'aide d'une méthode de descente basée sur les super-gradients pour la minimisation de fonctions sous-modulaires. Des résultats numériques ont été présentés, sur les mêmes instances que précédemment, avec toutefois des erreurs empiriques beaucoup plus importantes.

Perspectives

Nous présentons maintenant quelques perspectives de recherche que nous trouvons intéressantes au sujet de la résolution par élicitation incrémentale des préférences sur domaine combinatoire :

Autres problèmes d'optimisation combinatoire multi-objectifs : chacune des méthodes issues des travaux de cette thèse est potentiellement applicable à tout problème d'optimisation combinatoire multi-objectifs. Tester nos algorithmes sur de nombreux problèmes permettrait d'évaluer plus finement leurs performances. On peut citer en particulier le problème classique du clustering où l'objectif est le traitement de données afin de les regrouper selon une caractéristique commune [Anderberg, 2014]. Il existe un contexte où l'information préférentielle est également disponible sur ces données [Mukhopadhyay et al., 2015]. Le clustering étant une technique d'analyse exploratoire des données, il est par conséquent nécessaire d'utiliser des méthodes donnant de bons résultats sur des problèmes de grande taille, voire de très grande taille. Nous pourrions également tester nos algorithmes sur des problèmes et données issus du monde de l'industrie afin d'évaluer leurs performances dans des cas plus concret.

Autres heuristiques : en sortant du cadre des matroïdes, des travaux futurs peuvent être

effectués sur la base d'autres méthodes basées sur une recherche locale. On peut citer par exemple la recherche tabou [Glover, 1986] avec la laquelle nous avons déjà fait quelques tests préliminaires (non présentés dans cette thèse). Les résultats obtenus sur le problème de couverture maximale en minimisation sont très encourageants, cependant cet algorithme ne présente aucune garantie sur la qualité des résultats fournis. On peut aussi choisir d'utiliser d'autres méthodes de résolution issues de la recherche opérationnelle ou de l'intelligence artificielle. Une hybridation issue de la combinaison entre plusieurs métaheuristiques pourrait nous permettre d'avoir de meilleures performances, comme c'est le cas pour de nombreux problèmes d'optimisation combinatoire [Prins, 2009]. Il serait également intéressant d'étudier la configuration automatique des heuristiques développées dans cette thèse [Blot et al., 2019]. Par exemple, le paramètre delta, présent dans tous nos algorithmes, est un facteur important de nos méthodes, car il permet à la fois de limiter le nombre de questions et de diversifier la recherche (dans une fonction de voisinage cela permet de ne pas toujours sélectionner le meilleur des voisins et ainsi d'explorer de nouvelles zones de recherche). Dans nos méthodes, nous avons utilisé une valeur fixée en début d'algorithme, mais il serait intéressant de considérer des valeurs de delta pouvant varier en fonction de la progression de la méthode.

Stratégie de réduction des paramètres admissibles : dans tous les travaux de cette thèse, nous avons fait le choix de la CSS [Boutilier et al., 2006] basée sur le minimax regret [Savage, 1954]. C'est une stratégie largement employée et très documentée. Elle possède aussi d'autres avantages comme l'explicabilité. Cependant, nous pourrions comparer les résultats de nos algorithmes avec différentes stratégies de collecte des préférences du décideur. On peut également considérer le fait de poser des questions entre plus de deux solutions.

Borner le nombre de questions : nous avons observé que l'élicitation incrémentale permet de trouver la meilleure alternative possible avec, en moyenne, un nombre raisonnable de questions. Lorsque l'on autorise un seuil de tolérance sur la qualité de la solution retournée, ce nombre devient admissible pour le décideur. Les algorithmes interactifs présentés dans les chapitres 2 et 3 de cette thèse sont de complexité polynomiale et posent un nombre polynomial de questions. Cependant, aucune borne supérieure théorique sur le nombre de questions produites n'a été formulée pour la recherche locale interactive dans le cas sous-modulaire (4). L'obtention d'une telle borne permettrait d'acquérir une information supplémentaire sur le coût informationnel de nos algorithmes (coût de communication, complexité des questions).

Gestion des incohérences : les algorithmes de décision interactifs proposés reposent sur la réduction progressive de l'ensemble des fonctions admissibles jusqu'à ce qu'une solution optimale (ou quasi-optimale) puisse être identifiée. À chaque étape, les réductions de l'espace des paramètres sont dues aux nouvelles déclarations de préférences obtenues du décideur. Cela rend le processus de décision très sensible aux éventuelles erreurs dans les réponses. En effet, si le décideur se trompe à l'une des questions alors la recommandation faite ne sera pas forcément de bonne qualité. Une suite possible de ce travail serait de développer une approche plus flexible et tolérante aux erreurs dans la comparaison des solutions. Le défi serait de conserver la possibilité de fournir une bonne garantie sur la qualité de la solution finale tout en conservant des courtes séquences de questions de préférences dans le processus d'élicitation. Pour cela, une approche bayésienne pourrait être employée, comme dans l'article

[Bourdache et al., 2019]. Il faudrait l'adapter à des approches heuristiques.

Nos publications

Benabbou, N., Leroy, C., Lust, T., and Perny, P. (2021). Interactive optimization of submodular functions under matroid constraints. *In International Conference on Algorithmic Decision Theory ADT'21*, pages 307-322. Springer.

Benabbou, N., Leroy, C., Lust, T., and Perny, P. (2021). Combining preference elicitation with local search and greedy search for matroid optimization. *In 35th AAAI Conference on Artificial Intelligence AAAI'21.*

Benabbou, N., Leroy, C., and Lust, T. (2020a). An interactive regret-based genetic algorithm for solving multi-objective combinatorial optimization problems. *In Proceedings of the 34th AAAI Conference on Artificial Intelligence AAAI'20*, pages 2335-2342.

Benabbou, N., Leroy, C., and Lust, T. (2020). Regret-based elicitation for solving multi-objective knapsack problems with rank-dependent aggregators. *In 24th European Conference on Artificial Intelligence ECAI'20*, Santiago de Compostela, Spain.

Benabbou, N., Leroy, C., Lust, T., and Perny, P. (2019). Combining local search and elicitation for multi-objective combinatorial optimization. *In International Conference on Algorithmic Decision Theory ADT'19*, pages 1-16. Springer

Bibliographie

[Ahmed and Atamtürk, 2011] Ahmed, S. and Atamtürk, A. (2011). Maximizing a class of submodular utility functions. *Mathematical programming*, 128(1) :149–169.

[Anderberg, 2014] Anderberg, M. R. (2014). *Cluster analysis for applications : probability and mathematical statistics : a series of monographs and textbooks*, volume 19. Academic press.

[Angel et al., 2004] Angel, E., Bampis, E., and Gourvès, L. (2004). A dynasearch neighborhood for the bicriteria traveling salesman problem. In *Metaheuristics for Multiobjective Optimisation*, pages 153–176. Springer.

[Applegate et al., 2003] Applegate, D., Cook, W., and Rohe, A. (2003). Chained lin-kernighan for large traveling salesman problems. *INFORMS J. Comput.*, 15(1) :82–92.

[Arrow, 1950] Arrow, K. J. (1950). A difficulty in the concept of social welfare. *Journal of political economy*, 58(4) :328–346.

[Arrow, 2012] Arrow, K. J. (2012). *Social choice and individual values*. Yale university press.

[Bai and Bilmes, 2018] Bai, W. and Bilmes, J. A. (2018). Greed is still good : Maximizing monotone submodular+supermodular (BP) functions. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 314–323. PMLR.

[Bandyopadhyay et al., 2008] Bandyopadhyay, S., Saha, S., Maulik, U., and Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm : Amosa. *IEEE Transactions on Evolutionary Computation*, 12(3) :269–283.

[Basseur, 2006] Basseur, M. (2006). *Design of cooperative algorithms for multi-objective optimization : application to the flow-shop scheduling problem*. PhD thesis, Springer.

[Basseur et al., 2012] Basseur, M., Zeng, R.-Q., and Hao, J.-K. (2012). Hypervolume-based multi-objective local search. *Neural Computing and Applications*, 21(8) :1917–1929.

[Bazgan et al., 2009] Bazgan, C., Hugot, H., and Vanderpooten, D. (2009). Solving efficiently the 0-1 multi-objective knapsack problem. *Computers & OR*, 36(1) :260–279.

[Bellman, 1966] Bellman, R. (1966). Dynamic programming. *Science*, 153(3731) :34–37.

[Benabbou, 2017] Benabbou, N. (2017). *Procédures de décision par élicitation incrémentale de préférences en optimisation multicritère, multi-agents et dans l'incertain*. PhD thesis, Université Pierre et Marie Curie-Paris VI.

[Benabbou et al., 2016] Benabbou, N., Di Sabatino Di Diodoro, S., Perny, P., and Viappiani, P. (2016). Incremental preference elicitation in multi-attribute domains for choice and ranking with the Borda count. In *Proceedings of SUM'16*, pages 81–95.

[Benabbou et al., 2015] Benabbou, N., Gonzales, C., Perny, P., and Viappiani, P. (2015). Minimax regret approaches for preference elicitation with rank-dependent aggregators. *EURO journal on Decision processes*, 3(1) :29–64.

[Benabbou et al., 2020a] Benabbou, N., Leroy, C., and Lust, T. (2020a). An interactive regret-based genetic algorithm for solving multi-objective combinatorial optimization problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*, pages 2335–2342.

[Benabbou et al., 2020b] Benabbou, N., Leroy, C., and Lust, T. (2020b). Regret-based elicitation for solving multi-objective knapsack problems with rank-dependent aggregators. In *24th European Conference on Artificial Intelligence ECAI'20), Santiago de Compostela, Spain*.

[Benabbou et al., 2019] Benabbou, N., Leroy, C., Lust, T., and Perny, P. (2019). Combining local search and elicitation for multi-objective combinatorial optimization. In *International Conference on Algorithmic Decision Theory*, pages 1–16. Springer.

[Benabbou et al., 2021a] Benabbou, N., Leroy, C., Lust, T., and Perny, P. (2021a). Combining preference elicitation with local search and greedy search for matroid optimization. In *35th AAAI Conference on Artificial Intelligence (AAAI'21)*.

[Benabbou et al., 2021b] Benabbou, N., Leroy, C., Lust, T., and Perny, P. (2021b). Interactive optimization of submodular functions under matroid constraints. In *International Conference on Algorithmic Decision Theory*, pages 307–322. Springer.

[Benabbou and Lust, 2019a] Benabbou, N. and Lust, T. (2019a). A general interactive approach for solving multi-objective combinatorial optimization problems with imprecise preferences. In *Proceedings of SOCS'19*, pages 164–165.

[Benabbou and Lust, 2019b] Benabbou, N. and Lust, T. (2019b). An interactive polyhedral approach for multi-objective combinatorial optimization with incomplete preference information. In *International Conference on Scalable Uncertainty Management*, pages 221–235. Springer.

[Benabbou and Perny, 2015a] Benabbou, N. and Perny, P. (2015a). Combining preference elicitation and search in multiobjective state-space graphs. In *Proceedings of IJCAI'15*, pages 297–303.

[Benabbou and Perny, 2015b] Benabbou, N. and Perny, P. (2015b). Incremental weight elicitation for multiobjective state space search. In *Proceedings of AAAI'15*, pages 1093–1098.

[Benabbou and Perny, 2016] Benabbou, N. and Perny, P. (2016). Solving multi-agent knapsack problems using incremental approval voting. In *Proceedings of ECAI'16*, pages 1318–1326.

[Benabbou and Perny, 2017] Benabbou, N. and Perny, P. (2017). Adaptive elicitation of preferences under uncertainty in sequential decision making problems. In *Proceedings of IJCAI'17*, pages 4566–4572.

[Benabbou and Perny, 2018] Benabbou, N. and Perny, P. (2018). Interactive resolution of multiobjective combinatorial optimization problems by incremental elicitation of criteria weights. *EURO Journal on Decision Processes*, 6(3) :283–319.

[Benabbou et al., 2014] Benabbou, N., Perny, P., and Viappiani, P. (2014). Incremental elicitation of Choquet capacities for multicriteria decision making. In *Proceedings of ECAI'14*, pages 87–92.

[Benabbou et al., 2017] Benabbou, N., Perny, P., and Viappiani, P. (2017). Incremental elicitation of choquet capacities for multicriteria choice, ranking and sorting problems. *Artificial Intelligence*, 246 :152–180.

[Benayoun et al., 1971] Benayoun, R., De Montgolfier, J., Tergny, J., and Laritchev, O. (1971). Linear programming with multiple objective functions : Step method (stem). *Mathematical programming*, 1(1) :366–375.

[Bérubé et al., 2009] Bérubé, J.-F., Gendreau, M., and Potvin, J.-Y. (2009). An exact epsilon-constraint method for bi-objective combinatorial optimization problems : Application to the traveling salesman problem with profits. *European journal of operational research*, 194(1) :39–50.

[Blot et al., 2018] Blot, A., Marmion, M., and Jourdan, L. (2018). Survey and unification of local search techniques in metaheuristics for multi-objective combinatorial optimisation. *J. Heuristics*, 24(6) :853–877.

[Blot et al., 2019] Blot, A., Marmion, M., Jourdan, L., and Hoos, H. H. (2019). Automatic configuration of multi-objective local search algorithms for permutation problems. *Evol. Comput.*, 27(1) :147–171.

[Blum and Roli, 2003] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3) :268–308.

[Blum et al., 2008] Blum, C., Roli, A., and Sampels, M. (2008). *Hybrid metaheuristics : an emerging approach to optimization*, volume 114. Springer.

[Bourdache and Perny, 2019] Bourdache, N. and Perny, P. (2019). Algorithmes d'élicitation incrémentale des préférences pour la résolution de problèmes de sac-à-dos multi-agents équitables. In *ROADEF*.

[Bourdache et al., 2019] Bourdache, N., Perny, P., and Spanjaard, O. (2019). Incremental Elicitation of Rank-Dependent Aggregation Functions based on Bayesian Linear Regression. In *IJCAI-19*, pages 2023–2029, Macao, China.

[Bourdache et al., 2020] Bourdache, N., Perny, P., and Spanjaard, O. (2020). Bayesian preference elicitation for multiobjective combinatorial optimization. In *DA2PL 2020 - From Multiple Criteria Decision Aid to Preference Learning*, Trento, Italy.

[Boutilier et al., 2006] Boutilier, C., Patrascu, R., Poupart, P., and Schuurmans, D. (2006). Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artificial Intelligence*, 170(8-9) :686–713.

[Bowman, 1976] Bowman, V. J. (1976). On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In *Multiple criteria decision making*, pages 76–86. Springer.

[Branke et al., 2016] Branke, J., Corrente, S., Greco, S., Słowiński, R., and Zielniewicz, P. (2016). Using choquet integral as preference model in interactive evolutionary multiobjective optimization. *European Journal of Operational Research*, 250(3) :884–901.

[Branke and Deb, 2005] Branke, J. and Deb, K. (2005). *Integrating User Preferences into Evolutionary Multi-Objective Optimization*, pages 461–477. Knowledge Incorporation in Evolutionary Computation, Springer Berlin Heidelberg.

[Branke et al., 2010] Branke, J., Greco, S., Słowiński, R., and Zielniewicz, P. (2010). Interactive evolutionary multiobjective optimization driven by robust ordinal regression. *Bulletin of the Polish Academy of Sciences : Technical Sciences*.

[Bremermann, 1958] Bremermann, H. J. (1958). *The evolution of intelligence : The nervous system as a model of its environment*. University of Washington, Department of Mathematics.

[Caballero et al., 2007] Caballero, R., González, M., Guerrero, F. M., Molina, J., and Paralera, C. (2007). Solving a multiobjective location routing problem with a metaheuristic based on tabu search. application to a real case in andalusia. *European Journal of Operational Research*, 177(3) :1751–1763.

[Calinescu et al., 2011] Calinescu, G., Chekuri, C., Pal, M., and Vondrák, J. (2011). Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6) :1740–1766.

[Chateauneuf et al., 1999] Chateauneuf, A., Dana, R., and Tallon, J.-M. (1999). Diversification, convex preferences and non-empty core in the Choquet expected utility model. *Economic Theory*, 19(3) :509–523.

[Chateauneuf and Jaffray, 1989] Chateauneuf, A. and Jaffray, J.-Y. (1989). Some characterizations of lower probabilities and other monotone capacities through the use of Möbius inversion. *Mathematical Social Sciences*, 17(3) :263–283.

[Chlamtác et al., 2017] Chlamtác, E., Dinitz, M., and Makarychev, Y. (2017). Minimizing the union : Tight approximations for small set bipartite vertex expansion. In Klein, P. N., editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 881–899. SIAM.

[Choquet, 1953] Choquet, G. (1953). Theory of capacities. *Annales de l'Institut Fourier*, 5 :31–295.

[Cohen and Katzir, 2008] Cohen, R. and Katzir, L. (2008). The generalized maximum coverage problem. *Information Processing Letters*, 108(1) :15–22.

[Commander, 2009] Commander, C. W. (2009). *Maximum cut problem, MAX-CUTMaximum Cut Problem, MAX-CUT*, pages 1991–1999. Springer US, Boston, MA.

[Croes, 1958] Croes, G. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6) :791–812.

[Cunningham, 1985] Cunningham, W. H. (1985). On submodular function minimization. *Combinatorica*, 5 :185–192.

[Czyżżak and Jaszkiewicz, 1998] Czyżżak, P. and Jaszkiewicz, A. (1998). Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1) :34–47.

[Dächert et al., 2012] Dächert, K., Gorski, J., and Klamroth, K. (2012). An augmented weighted tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems. *Computers & Operations Research*, 39(12) :2929–2943.

[Dantzig et al., 1954] Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4) :393–410.

[Deb et al., 2000] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization : Nsga-ii. In *International conference on parallel problem solving from nature*, pages 849–858. Springer.

[Deb et al., 2002] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2) :182–197.

[Dempster, 2008] Dempster, A. P. (2008). Upper and lower probabilities induced by a multivalued mapping. In *Classic works of the Dempster-Shafer theory of belief functions*, pages 57–72. Springer.

[Dezső et al., 2011] Dezső, B., Jüttner, A., and Kovács, P. (2011). Lemon–an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5) :23–45.

[Diak et al., 1998] Diak, G. R., Anderson, M. C., Bland, W. L., Norman, J. M., Mecikalski, J. M., and Aune, R. M. (1998). Agricultural management decision aids driven by real-time satellite data. *Bulletin of the American Meteorological Society*, 79(7) :1345–1356.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1) :29–41.

[Drummond and Boutilier, 2013] Drummond, J. and Boutilier, C. (2013). Elicitation and approximately stable matching with partial preferences. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

[Dyer, 2005] Dyer, J. S. (2005). Maut—multiattribute utility theory. In *Multiple criteria decision analysis : state of the art surveys*, pages 265–292. Springer.

[Edmonds, 1971] Edmonds, J. (1971). Matroids and the greedy algorithm. *Mathematical programming*, 1(1) :127–136.

[Edmonds and Fulkerson, 1965] Edmonds, J. and Fulkerson, D. R. (1965). Transversals and matroid partition. *J. Res. Nat. Bur. Standards Sect. B*, 69 :147–153.

[E.H. Aarts, 2003] E.H. Aarts, J. L. (2003). *Local search in combinatorial optimization*. Princeton University Press.

[Ehrgott, 2005] Ehrgott, M. (2005). *Multicriteria Optimization. Second edition.* Springer, Berlin.

[Filmus and Ward, 2012] Filmus, Y. and Ward, J. (2012). The power of local search : Maximum coverage over a matroid. In *STACS'12 (29th Symposium on Theoretical Aspects of Computer Science)*, volume 14, pages 601–612. LIPIcs.

[Fisher et al., 1978] Fisher, M., Nemhauser, G., and Wolsey, L. (1978). *An analysis of approximations for maximizing submodular set functions-II*, volume 8, pages 73–87. Springer.

[Fogel and Anderson, 2000] Fogel, D. B. and Anderson, R. W. (2000). Revisiting bremermann's genetic algorithm. i. simultaneous mutation of all parameters. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*, volume 2, pages 1204–1209. IEEE.

[Fogel et al., 1966] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). Intelligent decision making through a simulation of evolution. *Behavioral science*, 11(4) :253–272.

[Gandibleux et al., 1997] Gandibleux, X., Mezdaoui, N., and Fréville, A. (1997). A tabu search procedure to solve multiobjective combinatorial optimization problems. In *Advances in multiple objective and goal programming*, pages 291–300. Springer.

[Garey, 1979] Garey, M. R. (1979). A guide to the theory of np-completeness. *Computers and intractability.*

[Gendreau, 2003] Gendreau, M. (2003). An introduction to tabu search. In *Handbook of metaheuristics*, pages 37–54. Springer.

[Geoffrion, 1968] Geoffrion, A. M. (1968). Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22(3).

[Giard and Roy, 1985] Giard, V. E. and Roy, B. (1985). *Méthodologie multicritère d'aide à la décision.* Editions Economica.

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5) :533–549.

[Glover and Sörensen, 2015] Glover, F. and Sörensen, K. (2015). Metaheuristics. *Scholarpedia*, 10(4) :6532.

[Glover and Kochenberger, 2006] Glover, F. W. and Kochenberger, G. A. (2006). *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.

[Godart, 2001] Godart, J. (2001). *Problèmes d'optimisation combinatoire à caractère économique dans le secteur du tourisme (organisation de voyages).* PhD thesis, PhD thesis, Faculté Warocqué des sciences économiques, Université de Mons.

[Goldberg et al., 1989] Goldberg, R. B., Barker, S. J., and Perez-Grau, L. (1989). Regulation of gene expression during plant embryogenesis. *Cell*, 56(2) :149–160.

[Goldsmith et al., 2014] Goldsmith, J., Lang, J., Mattei, N., and Perny, P. (2014). Voting with rank dependent scoring rules. In *Proceedings of AAAI'14*, pages 698–704.

[Grabisch, 2005] Grabisch, M. (2005). Une approche constructive de la décision multicritère. *Traitement du signal*, 22(4) :321–337.

[Grabisch and Labreuche, 2010] Grabisch, M. and Labreuche, C. (2010). A decade of application of the Choquet and Sugeno integrals in multi-criteria decision aid. *Annals of Operations Research*, 175(1) :247–286.

[Grabisch et al., 2009] Grabisch, M., Marichal, J.-L., Mesiar, R., and Pap, E. (2009). *Aggregation Functions*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, New-York.

[Grabisch, 2003] Grabisch, M. et Perny, P. (2003). Logique floue, principes, aide à la décision : chapitre agrégation multicritère", traité ic2, série informatique et systèmes d'information.

[Greene and Magnanti, 1975] Greene, C. and Magnanti, T. L. (1975). Some abstract pivot algorithms. *SIAM Journal on Applied Mathematics*, 29(3) :530–539.

[Grefenstette, 1993] Grefenstette, J. J. (1993). Genetic algorithms and machine learning. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 3–4.

[Haimes, 1971] Haimes, Y. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, 1(3) :296–297.

[Hansen, 2000] Hansen, M. (2000). Tabu search for multiobjective combinatorial optimization : Tamoco. *Control and Cybernetics*, 29(3) :799–818.

[Hansen et al., 1997] Hansen, M. P. et al. (1997). Tabu search for multiobjective optimization : Mots. In *Proceedings of the 13th international conference on multiple criteria decision making*, pages 574–586. Citeseer.

[Hochbaum and Pathria, 1998] Hochbaum, D. S. and Pathria, A. (1998). Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics (NRL)*, 45(6) :615–627.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[Huédé et al., 2006] Huédé, F. L., Grabisch, M., Labreuche, C., and Savéant, P. (2006). Mcs—a new algorithm for multicriteria optimisation in constraint programming. *Annals of Operations Research*, 147(1) :143–174.

[Iyer et al., 2013] Iyer, R. K., Jegelka, S., and Bilmes, J. A. (2013). Fast semidifferential-based submodular function optimization. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 855–863. JMLR.org.

[Jacquet-Lagrèze and Siskos, 1982] Jacquet-Lagrèze, E. and Siskos, J. (1982). Assessing a set of additive utility functions for multicriteria decision-making, the UTA method. *European journal of operational research*, 10(2) :151–164.

[Jaszkiewicz, 2018] Jaszkiewicz, A. (2018). Many-objective Pareto local search. *European Journal of Operational Research*, 271(3) :1001–1013.

[Jaszkiewicz and Lust, 2018] Jaszkiewicz, A. and Lust, T. (2018). ND-Tree-based update : A fast algorithm for the dynamic nondominance problem. *IEEE Trans. Evolutionary Computation*, 22(5) :778–791.

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

[Jozefowiez et al., 2005] Jozefowiez, N., Semet, F., and Talbi, E.-G. (2005). Enhancements of nsga ii and its application to the vehicle routing problem with route balancing. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 131–142. Springer.

[Kaddani et al., 2017] Kaddani, S., Vanderpooten, D., Vanpeperstraete, J.-M., and Aissi, H. (2017). Weighted sum model with partial preference information : application to multi-objective optimization. *EJOR*, 260 :665–679.

[Kannan et al., 2008] Kannan, S., Baskar, S., McCalley, J. D., and Murugan, P. (2008). Application of nsga-ii algorithm to generation expansion planning. *IEEE Transactions on Power systems*, 24(1) :454–461.

[Karasakal and Köksalan, 2009] Karasakal, E. and Köksalan, M. (2009). Generating a representative subset of the nondominated frontier in multiple criteria decision making. *Operations Research*, 57(1) :187–199.

[Keeney et al., 1993] Keeney, R. L., Raiffa, H., and Meyer, R. F. (1993). *Decisions with multiple objectives : preferences and value trade-offs*. Cambridge university press.

[Khuller et al., 1999] Khuller, S., Moss, A., and Naor, J. S. (1999). The budgeted maximum coverage problem. *Information processing letters*, 70(1) :39–45.

[Kirkpatrick, 1984] Kirkpatrick, S. (1984). Optimization by simulated annealing : Quantitative studies. *Journal of statistical physics*, 34(5) :975–986.

[Kirlik and Sayın, 2014] Kirlik, G. and Sayın, S. (2014). A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3) :479–488.

[Kiziltan and Yucaoğlu, 1983] Kiziltan, G. and Yucaoğlu, E. (1983). An algorithm for multiobjective zero-one linear programming. *Management Science*, 29(12) :1444–1453.

[Klamroth and Wiecek, 2000] Klamroth, K. and Wiecek, M. M. (2000). Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics (NRL)*, 47(1) :57–76.

[Korhonen, 2005] Korhonen, P. (2005). *Multiple criteria decision analysis*. Greco, Salvatore and Figueira, J and Ehrgott, M. Springer.

[Korhonen et al., 1990] Korhonen, P., Moskowitz, H., and Wallenius, J. (1990). Choice behavior in interactive multiple-criteria decision making. *Annals of Operations Research*, 23(1) :161–179.

[Kouvelis and Yu, 2013] Kouvelis, P. and Yu, G. (2013). *Robust discrete optimization and its applications*, volume 14. Springer Science & Business Media.

[Koza, 1992] Koza, J. (1992). Genetic programming : on the programming of computers by means of natural selection cambridge. *MA : MIT Press.[Google Scholar]*.

[Kruskal, 1956] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *American Mathematical Society*, 7 :48–50.

[Land and Doig, 2010] Land, A. H. and Doig, A. G. (2010). An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer.

[Laumanns et al., 2006] Laumanns, M., Thiele, L., and Zitzler, E. (2006). An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3) :932–942.

[Lee and Pulat, 1993] Lee, H. and Pulat, P. S. (1993). Bicriteria network flow problems : Integer case. *European Journal of Operational Research*, 66(1) :148–157.

[Lee, 2004] Lee, J. (2004). *A first course in combinatorial optimization*, volume 36. Cambridge University Press.

[Lehmann et al., 2006] Lehmann, B., Lehmann, D., and Nisan, N. (2006). Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2) :270–296.

[Leitner et al., 2015] Leitner, M., Ljubić, I., and Sinnl, M. (2015). A computational study of exact approaches for the bi-objective prize-collecting steiner tree problem. *INFORMS Journal on Computing*, 27(1) :118–134.

[Lin and Kernighan, 1973] Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21(2) :498–516.

[Lovász, 1983] Lovász, L. (1983). Submodular functions and convexity. In *Mathematical Programming, the State of the Art*, pages 235–257. A. Bachem and M. Grötschel and B. Korte.

[Mareschal et al., 1984] Mareschal, B., Brans, J. P., Vincke, P., et al. (1984). Promethee : A new family of outranking methods in multicriteria analysis. Technical report, ULB–Universite Libre de Bruxelles.

[Martello and Toth, 1990] Martello, S. and Toth, P. (1990). *Knapsack problems : algorithms and computer implementations*. John Wiley & Sons, Inc.

[Martins, 1984] Martins, E. Q. V. (1984). On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2) :236–245.

[McCormick, 2005] McCormick, S. T. (2005). Submodular function minimization. In Aardal, K., Nemhauser, G., and Weismantel, R., editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 321–391. Elsevier.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6) :1087–1092.

[Miller et al., 1995] Miller, B. L., Goldberg, D. E., et al. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3) :193–212.

[Mukhopadhyay et al., 2015] Mukhopadhyay, A., Maulik, U., and Bandyopadhyay, S. (2015). A survey of multiobjective evolutionary clustering. *ACM Comput. Surv.*, 47(4).

[Nemhauser et al., 1978] Nemhauser, G., Wolsey, L., and Fisher, M. (1978). An analysis of approximations for maximizing submodular set functions-I. *Mathematical programming*, 14(1) :265–294.

[O'connor, 2001] O'connor, A. (2001). Using patient decision aids to promote evidence-based decision making. *BMJ Evidence-Based Medicine*, 6(4) :100–102.

[Ogryczak and Śliwiński, 2003] Ogryczak, W. and Śliwiński, T. (2003). On solving linear programs with the ordered weighted averaging objective. *European Journal of Operational Research*, 148(1) :80–91.

[Olson, 1997] Olson, D. L. (1997). Decision aids for selection problems. *Journal of the Operational Research Society*, 48(5) :541–542.

[Oxley, 2006] Oxley, J. G. (2006). *Matroid theory*, volume 3. Oxford University Press, USA.

[Özlen and Azizoğlu, 2009] Özlen, M. and Azizoğlu, M. (2009). Multi-objective integer programming : A general approach for generating all non-dominated solutions. *European Journal of Operational Research*, 199(1) :25–35.

[Özpeynirci and Köksalan, 2010] Özpeynirci, Ö. and Köksalan, M. (2010). An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, 56(12) :2302–2315.

[Paquete et al., 2004] Paquete, L., Chiarandini, M., and Stützle, T. (2004). Pareto local optimum sets in the biobjective traveling salesman problem : An experimental study. In *Metaheuristics for multiobjective optimisation*, pages 177–199. Springer.

[Pedersen et al., 2008] Pedersen, C. R., Nielsen, L. R., and Andersen, K. A. (2008). The bicriterion multimodal assignment problem : Introduction, analysis, and experimental results. *INFORMS Journal on Computing*, 20(3) :400–411.

[Perny, 2000] Perny, P. (2000). Modélisation des préférences, agrégation multicritere et systemes d'aidea la décision. *These d'habilitation, Université Pierre et Marie Curie, Paris.*

[Pounds, 1965] Pounds, W. F. (1965). *The process of problem finding.* [Cambridge, Mass., MIT].

[Prins, 2009] Prins, C. (2009). A grasp× evolutionary local search hybrid for the vehicle routing problem. In *Bio-inspired algorithms for the vehicle routing problem*, pages 35–53. Springer.

[Przybylski and Gandibleux, 2017] Przybylski, A. and Gandibleux, X. (2017). Multi-objective branch and bound. *European Journal of Operational Research*, 260(3) :856–872.

[Przybylski et al., 2008] Przybylski, A., Gandibleux, X., and Ehrgott, M. (2008). Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, 185(2) :509–533.

[Przybylski et al., 2010a] Przybylski, A., Gandibleux, X., and Ehrgott, M. (2010a). A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *INFORMS Journal on Computing*, 22(3) :371–386.

[Przybylski et al., 2010b] Przybylski, A., Gandibleux, X., and Ehrgott, M. (2010b). A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization*, 7(3).

[Raith and Ehrgott, 2009a] Raith, A. and Ehrgott, M. (2009a). A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4) :1299–1331.

[Raith and Ehrgott, 2009b] Raith, A. and Ehrgott, M. (2009b). A two-phase algorithm for the biobjective integer minimum cost flow problem. *Computers & Operations Research*, 36(6) :1945–1954.

[Resende, 1998] Resende, M. (1998). Computing approximate solutions of the maximum covering problem with GRASP. *Journal of Heuristics*, 4(2) :161–177.

[Roughgarden, 2017] Roughgarden, T. (2017). *Algorithms Illuminated*. Soundlikeyourself publishing.

[Roy, 1968] Roy, B. (1968). Classement et choix en présence de points de vue multiples. *Revue française d'informatique et de recherche opérationnelle*, 2(8) :57–75.

[Roy, 1996] Roy, B. (1996). *Multicriteria methodology for decision aiding*, volume 12. Springer Science & Business Media.

[Roy, 2005] Roy, B. (2005). Paradigms and challenges. In *Multiple criteria decision analysis : state of the art surveys*, pages 3–24. Springer.

[Rubenstein and Haberstroh, 1960] Rubenstein, A. H. and Haberstroh, C. J. (1960). *Some theories of organization*. Dorsey Press.

[Russell and Norvig, 2002] Russell, S. and Norvig, P. (2002). *Artificial intelligence : a modern approach*. Prentice Hall Press.

[Sampson, 1976] Sampson, J. R. (1976). Adaptation in natural and artificial systems (john h. holland).

[Savage, 1954] Savage, L. J. (1954). The foundations of statistics ; jon wiley and sons. *Inc. : New York, NY, USA*.

[Savage, 1972] Savage, L. J. (1972). *The foundations of statistics*. Courier Corporation.

[Schmeidler, 1986] Schmeidler, D. (1986). Integral representation without additivity. *Proceedings of the American Mathematical Society*, 97(2) :255–261.

[Schrijver, 2000] Schrijver, A. (2000). A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2) :346–355.

[Schwefel, 1981] Schwefel, H.-P. (1981). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

[Sedeno-Noda and González-Martın, 2001] Sedeno-Noda, A. and González-Martın, C. (2001). An algorithm for the biobjective integer minimum cost flow problem. *Computers & Operations Research*, 28(2) :139–156.

[Shafer, 1976] Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

[Simon, 1960] Simon, H. A. (1960). *The new science of management decision.* Harper & Brothers.

[Siskos and Yannacopoulos, 1985] Siskos, Y. and Yannacopoulos, D. (1985). UTASTAR : An ordinal regression method for building additive value functions. *Investigação Operacional*, 5(1) :39–53.

[Skowron, 2017] Skowron, P. (2017). FPT approximation schemes for maximizing submodular functions. *Information and Computation*, 257 :65–78.

[Skowron et al., 2016] Skowron, P., Faliszewski, P., and Lang, J. (2016). Finding a collective set of items : From proportional multirepresentation to group recommendation. *Artif. Intell.*, 241 :191–216.

[Sörensen and Glover, 2013] Sörensen, K. and Glover, F. (2013). Metaheuristics. *Encyclopedia of operations research and management science*, 62 :960–970.

[Sourd and Spanjaard, 2008] Sourd, F. and Spanjaard, O. (2008). A multiobjective branch-and-bound framework : Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3) :472–484.

[Soyel et al., 2011] Soyel, H., Tekguc, U., and Demirel, H. (2011). Application of nsga-ii to feature selection for facial expression recognition. *Computers & Electrical Engineering*, 37(6) :1232–1240.

[Srinivas and Deb, 1994] Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3) :221–248.

[Steiner and Radzik, 2008] Steiner, S. and Radzik, T. (2008). Computing all efficient solutions of the biobjective minimum spanning tree problem. *Computers & Operations Research*, 35(1) :198–211.

[Steuer, 1986] Steuer, R. E. (1986). Multiple criteria optimization. *Theory, computation and applications.*

[Steuer and Choo, 1983] Steuer, R. E. and Choo, E.-U. (1983). An interactive weighted tchebycheff procedure for multiple objective programming. *Mathematical programming*, 26(3) :326–344.

[Steuer et al., 2005] Steuer, R. E., Qi, Y., and Hirschberger, M. (2005). Multiple objectives in portfolio selection. *Journal of financial decision making*, 1(1) :5–20.

[Stoer and Wagner, 1997] Stoer, M. and Wagner, F. (1997). A simple min-cut algorithm. *J. ACM*, 44(4) :585–591.

[Suman and Kumar, 2006] Suman, B. and Kumar, P. (2006). A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the operational research society*, 57(10) :1143–1160.

[Teghem, 2003] Teghem, J. (2003). *Programmation linéaire.* Paris.

[Teghem and Pirlot, 2002] Teghem, J. and Pirlot, M. (2002). *Optimisation approchée en recherche opérationnelle : recherches locales, réseaux neuronaux et satisfaction de contraintes*. Hermes Science Publications.

[Teng and Tzeng, 1996] Teng, J.-Y. and Tzeng, G.-H. (1996). A multiobjective programming approach for selecting non-independent transportation investment alternatives. *Transportation Research Part B : Methodological*, 30(4) :291–307.

[Torra, 1997] Torra, V. (1997). The weighted OWA operator. *International Journal of Intelligent Systems*, 12(2) :153–166.

[Ulungu and Teghem, 1995] Ulungu, E. L. and Teghem, J. (1995). The two phases method : An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of computing and decision sciences*, 20(2) :149–165.

[Ulungu et al., 1999] Ulungu, E. L., Teghem, J., Fortemps, P., and Tuyttens, D. (1999). Mosa method : a tool for solving multiobjective combinatorial optimization problems. *Journal of multicriteria decision analysis*, 8(4) :221.

[Vaidya and Kumar, 2006] Vaidya, O. S. and Kumar, S. (2006). Analytic hierarchy process : An overview of applications. *European Journal of operational research*, 169(1) :1–29.

[Vanderpooten and Vincke, 1989] Vanderpooten, D. and Vincke, P. (1989). Description and analysis of some representative interactive multicriteria procedures. In *Models and Methods in Multiple Criteria Decision Making*, pages 1221–1238. Elsevier.

[Verma et al., 2021] Verma, S., Pant, M., and Snasel, V. (2021). A comprehensive review on nsga-ii for multi-objective combinatorial optimization problems. *IEEE Access*, 9 :57757–57791.

[Vincent et al., 2013] Vincent, T., Seipp, F., Ruzika, S., Przybylski, A., and Gandibleux, X. (2013). Multiple objective branch and bound for mixed 0-1 linear programming : Corrections and improvements for the biobjective case. *Computers & Operations Research*, 40(1) :498–509.

[Visée et al., 1998] Visée, M., Teghem, J., Pirlot, M., and Ulungu, E. L. (1998). Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, 12(2) :139–155.

[Von Neumann and Morgenstern, 1947] Von Neumann, J. and Morgenstern, O. (1947). *Theory of games and economic behavior, 2nd rev*. Princeton university press.

[Vondrák, 2008] Vondrák, J. (2008). Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 67–74.

[Wang et al., 2019] Wang, S., Zhao, D., Yuan, J., Li, H., and Gao, Y. (2019). Application of nsga-ii algorithm for fault diagnosis in power system. *Electric Power Systems Research*, 175 :105893.

[Weymark, 1981] Weymark, J. (1981). Generalized Gini inequality indices. *Mathematical Social Sciences*, 1(4) :409–430.

[White et al., 1984] White, C. C., Sage, A. P., and Dozono, S. (1984). A model of multiattribute decisionmaking and trade-off weight determination under uncertainty. *IEEE Transactions on Systems, Man, and Cybernetics*, 14(2) :223–229.

[Whitney, 1992] Whitney, H. (1992). On the abstract properties of linear dependence. In *Hassler Whitney Collected Papers*, pages 147–171. Springer.

[Wiecek and Hadavas, 1997] Wiecek, M. M. and Hadavas, P. T. (1997). A tchebycheff metric approach to the optimal path problem with nonlinear multiattribute cost functions. In *Multiple Criteria Decision Making*, pages 445–454. Springer.

[Wierzbicki, 1986a] Wierzbicki, A. P. (1986a). On the completeness and constructiveness of parametric characterizations to vector optimization problems. *Operations-Research-Spektrum*, 8(2) :73–87.

[Wierzbicki, 1986b] Wierzbicki, A. P. (1986b). On the completeness and constructiveness of parametric characterizations to vector optimization problems. *Operations-Research-Spektrum*, 8(2) :73–87.

[Wolf and Merz, 2007] Wolf, S. and Merz, P. (2007). Evolutionary local search for the super-peer selection problem and the p-hub median problem. In *International workshop on hybrid metaheuristics*, pages 1–15. Springer.

[Yager, 1988] Yager, R. (1988). On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. Syst. Man Cybern.*, 18(1) :183–190.

[Yang et al., 2014] Yang, X.-S., Deb, S., and Fong, S. (2014). Metaheuristic algorithms : optimal balance of intensification and diversification. *Applied Mathematics & Information Sciences*, 8(3) :977.

[Yao et al., 1999] Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2) :82–102.

[Yin Kwong et al., 2014] Yin Kwong, W., Yun Zhang, P., Romero, D., Moran, J., Morgenroth, M., and Amon, C. (2014). Multi-objective wind farm layout optimization considering energy generation and noise propagation with nsga-ii. *Journal of Mechanical Design*, 136(9) :091010.