UNIVERSITÉ DE MONTRÉAL

MÉTAHEURISTIQUES HYBRIDES POUR LES PROBLÈMES DE RECOUVREMENT ET RECOUVREMENT PARTIEL D'ENSEMBLES APPLIQUÉS AU PROBLÈME DE POSITIONNEMENT DES TROUS DE FORAGE DANS LES MINES

NEHMÉ BILAL
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
AOÛT 2014

© Nehmé Bilal, 2014.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

MÉTAHEURISTIQUES HYBRIDES POUR LES PROBLÈMES DE RECOUVREMENT ET RECOUVREMENT PARTIEL D'ENSEMBLES APPLIQUÉS AU PROBLÈME DE POSITIONNEMENT DES TROUS DE FORAGE DANS LES MINES

présentée par : BILAL Nehmé
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. PESANT Gilles, Ph.D., président
M. GUIBAULT François, Ph.D., membre et directeur de recherche
M. GALINIER Philippe, Doct., membre et codirecteur de recherche
M. GAMACHE Michel, Ph.D., membre
M. QUIMPER Claude-Guy, Ph.D., membre

À ma mère ...

REMERCIEMENTS

Je tiens à remercier d'abord mes directeurs de recherche, Francois Guibault et Philippe Galinier, pour leur support précieux, leur implication et leur disponibilité tout au long de mon doctorat.

Un remerciement spécial à mon directeur de recherche, Francois Guibault, pour ses conseils, sa flexibilité et sa confiance qui m'ont permis de réaliser mon doctorat dans des conditions exceptionnelles. Grâce à Francois, j'ai eu la chance d'effectuer ma recherche en entreprise. De plus, la collaboration entre Francois et l'entreprise Objectivity m'a permis d'avoir un financement très convenable durant mon doctorat.

Je remercie aussi grandement mon codirecteur de recherche, Philippe Galinier, pour son implication très précieuse dans mon travail de recherche. L'expertise de Philippe en algorithmes d'optimisation a joué un rôle très important dans le succès de mon travail de recherche.

Un autre remerciement spécial est adressé à mon superviseur en entreprise, Andrew Dasys, aussi pour sa confiance et sa flexibilité qui m'ont permis de retourner à l'école Polytechnique de Montréal pour faire un doctorat soutenu par un projet industriel.

Finalement, je remercie Marie-Gabrielle Vallet pour son aide très utile dans plusieurs étapes de ce projet et Chris Cameron pour avoir été le premier à penser à la modélisation du problème de positionnement des trous de forage dans les mines sous forme d'un problème de recouvrement d'ensembles.

RÉSUMÉ

La première étape du cycle minier est l'exploration minérale. Dans cette étape, des longs trous de forage sont forés dans les zones de minéralisation pour extraire des échantillons. Les échantillons sont ensuite analysés et un modèle 3D de la distribution des minéraux dans la mine est construit. Puisque le forage coûte très cher, les géologues et ingénieurs miniers tentent de positionner leurs trous d'une façon qui minimise le coût de forage. Par contre, les techniques courantes utilisées pour minimiser le coût de forage sont peu sophistiquées et ne trouvent généralement pas la solution optimale.

Dans cette thèse, nous utilisons des techniques de recherche opérationnelle pour résoudre le problème de positionnement des trous de forage dans les mines. Nous modélisons le problème sous forme d'une variante du problème de recouvrement d'ensembles, qui est un problème très populaire en recherche opérationnelle, et résolvons ce problème à l'aide d'algorithmes métaheuristiques, notamment l'algorithme génétique, la recherche locale itérée et la recherche taboue.

Pour évaluer l'efficacité de notre approche, nous comparons les solutions trouvées par notre approche aux solutions trouvées par les approches industrielles sur des problèmes réels. Les résultats obtenus montrent que notre approche permet une réduction des coûts de forage allant jusqu'à 35%.

Un autre aspect très important de cette thèse est la résolution du problème de recouvrement d'ensembles (SCP) à l'aide de métaheuristiques. Nous proposons une nouvelle formulation du SCP et un nouvel algorithme pour le résoudre. La nouvelle formulation élimine les problèmes de faisabilité et redondances du SCP. Nos expérimentations ont montré que l'algorithme proposé trouve des meilleurs résultats que la majorité (si pas tous) les algorithmes métaheuristiques existants pour le SCP.

ABSTRACT

The first steps in the mining cycle are exploration and feasibility. In the exploration stage, geologists start by estimating the potential locations of mineral deposits. Then, they drill many long holes inside the mine to extract samples. The samples are then analyzed and a 3D model representing the distribution of mineralization in the mine is constructed. Because drilling is expensive, geologists and mining engineers try to position their drill holes to cover most potential sites with a minimum amount of drilling. However, the current techniques used to position the drill holes are inefficient and do not generally find the optimal solution.

In this thesis, we use operations research techniques to solve the drill holes placement problem. We model the drill holes placement problem as a variant of the set covering problem (which is a very popular optimization problem) and solve the modelled problem using the combination of multiple metaheuristic algorithms, namely the genetic algorithm, iterated local search and tabu search.

To evaluate the effectiveness of our approach, we compare the solutions found using our approach to the solutions found by industrial approaches on real world problems. The obtained results show that our approach allow saving up to 35% of drilling cost.

Another primary aspect of the thesis is the resolution of the set covering problem (SCP) using metaheuristic approaches. We propose a new formulation of the SCP and a new metaheuristic algorithm to solve it. The new formulation is specially designed for metaheuristic approaches and allows solving the SCP without having to deal with feasibility and set redundancy. Computational results show that our metaheuristic approach is more effective than most (if not all) metaheuristic approaches for the SCP.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

SCP	Set Covering Problem
USCP	Unicost Set Covering Problem
PTF	Positionnement des trous de forage
PMSCP	Profit Maximization Set Covering Problem
ITS	Iterated-Tabu-Search
BITS	Budgeted Iterated-Tabu-Search
GATS	Genetic-Tabu

CHAPITRE 1

INTRODUCTION

Les techniques de recherche opérationnelle sont de plus en plus utilisées pour offrir des outils d'aide à la décision à l'industrie moderne. Ces techniques permettent de modéliser et résoudre des problèmes d'optimisation difficiles qui ne peuvent être résolus à la main ou à l'aide d'un simple algorithme d'énumération. L'objet est habituellement d'améliorer l'efficacité ou de réduire le coût d'un processus quelconque.

Au cours des années, certains problèmes d'optimisation classiques ont été identifiés et utilisés pour modéliser un grand nombre d'applications industrielles. Cette réutilisation de modèles (problèmes classiques) permet aussi la réutilisation des algorithmes existants, qui ont été développés pour résoudre ces problèmes typiques, pour traiter des nouvelles applications. Parmi les problèmes classiques, on trouve le problème du voyageur de commerce, le problème de coloriage de graphe, le problème du sac à dos, le problème de recouvrement d'ensembles et autres.

Parce que les algorithmes développés pour résoudre les problèmes classiques sont réutilisés dans beaucoup d'applications, beaucoup de chercheurs se sont intéressés à résoudre ces problèmes. Plusieurs approches ont été utilisées dans la littérature dont les méthodes exactes, les algorithmes d'approximations et les méthodes heuristiques. Les méthodes exactes permettent d'obtenir une solution dont l'optimalité est garantie. Par contre, pour les problèmes NP-difficiles [42], dont la complexité est exponentielle, le temps requis pour trouver la solution optimale ou une solution de bonne qualité à l'aide d'une méthode exacte peut être très long (surtout pour les problèmes de grande taille). Dans de tels cas, on a habituellement recours aux algorithmes d'approximation ou aux méthodes heuristiques. Ces deux approches ne garantissent pas l'optimalité des solutions trouvées, mais permettent habituellement de trouver des solutions de bonne qualité dans un temps raisonnable. La différence principale entre une méthode heuristique et un algorithme d'approximation est le fait qu'un algorithme d'approximation offre une garantie quant à la qualité de la solution tandis qu'une méthode heuristique n'offre aucune telle garantie. Une métaheuristique est un algorithme heuristique général qui peut s'appliquer à différents problèmes d'optimisation. Parmi les métaheuristiques populaires, on trouve l'algorithme génétique, le recuit simulé, la recherche taboue, etc.

Dans cette thèse, nous nous intéressons au développement d'algorithmes métaheuristiques hybrides pour résoudre les problèmes de recouvrement et recouvrement partiel d'ensembles. Nous modélisons le problème de positionnement des trous de forage dans les mines (PTF)

à l'aide d'une nouvelle variante du problème de recouvrement partiel d'ensembles et développons une métaheuristique hybride pour résoudre le problème modélisé. De plus, nous proposons une nouvelle formulation mathématique du problème de recouvrement d'ensembles (SCP), qui est mieux adaptée pour la résolution de ce dernier à l'aide de métaheuristiques, et un nouvel algorithme métaheuristique efficace pour résoudre cette formulation. Cet algorithme combine l'algorithme génétique et la recherche taboue et trouve des meilleurs résultats que les algorithmes métaheuristiques existants pour le SCP.

La thèse est organisée comme suit. Dans ce chapitre, le problème de positionnement des trous de forage dans les mines est décrit et les objectifs du projet sont définis. Dans le chapitre 2, les techniques courantes de positionnement des trous de forage dans les mines et les différentes variantes du SCP sont revues. Au chapitre 3, la démarche de l'ensemble du travail et la cohérence des articles par rapport aux objectifs du projet sont discutées. Les articles de revues rédigés dans cette thèse sont présentés aux chapitres 4, 5 et 6. Au chapitre 7, l'approche utilisée pour prendre en compte la contrainte de budget dans le problème PTF est discutée. Au chapitre 8, certaines techniques de pré-conditionnement du problème PTF qui permettent d'adapter notre approche à plusieurs scénarios d'utilisation courants sont discutées. Quatres cas tests réels sont présentés au chapitre 9. La thèse est terminée par une discussion générale (chapitre 10) et une conclusion (chapitre 11).

1.1 Définition du problème

La première étape du cycle minier est l'exploration minérale. Cette étape vise à localiser et définir un site de minéralisation économiquement exploitable. L'exploration minérale peut être divisée en quatre étapes : inspection du terrain, création de cartes géologiques, forage d'exploration et forage de définition. À la première étape, les géologues identifient les sites de minéralisation potentiels en effectuant des analyses géologiques, géochimiques et géophysiques sur plusieurs sites de minéralisation. Pour faire leurs analyses, les géologues ont recours à des images à haute résolution des sites de minéralisation et effectuent des visites physiques aux régions prometteuses pour prélever des échantillons. Suite à l'identification d'un site de minéralisation potentiel, des cartes géologiques sont construites pour caractériser les zones de minéralisation. Ces cartes sont utilisées pour planifier le forage d'exploration. Le forage d'exploration consiste à forer un nombre limité de trous de forage pour explorer les zones de minéralisation identifiées dans les cartes géologiques. Les trous d'exploration sont des trous préliminaires largement espacés qui visent à extraire un premier échantillon de chaque zone cible. Suite au forage d'exploration, les géologues subdivisent les zones cibles sous forme de blocs tridimensionnels et estiment la densité de minerai dans chaque bloc en effectuant des

interpolations des données. Le résultat est un modèle 3D de blocs. Puisque les valeurs de densité de minerai attribuées aux blocs sont calculées par interpolation, ces valeurs sont sujettes à des incertitudes. Les valeurs des incertitudes sont calculées en utilisant une technique appelée le Krigeage [91]. Suite au calcul des incertitudes, les blocs sont classés sous trois catégories :

— *Blocs inférés* : Les blocs à incertitude élevée dont le niveau de confiance dans les estimations n'est pas suffisant pour permettre une analyse économique. Géométriquement, ces blocs se trouvent à une distance élevée des trous d'exploration.

— *Blocs indiqués* : Les blocs à incertitude relativement basse dont le niveau de confiance dans les estimations est raisonnable pour permettre une analyse économique. Géométriquement, ces blocs se trouvent à proximité (mais pas très proches) d'un ou plusieurs trous d'exploration.

— *Blocs mesurés* : Les blocs à basse incertitude dont le niveau de confiance dans les estimations est élevé pour permettre une analyse économique. Ces blocs se trouvent dans le voisinage immédiat de un ou plusieurs trous de forage.

Avant de passer à l'exploitation minière, la majorité des blocs doivent être indiqués ou mesurés pour minimiser les risques (idéalement, tous les blocs doivent être mesurés, mais à cause du budget limité, les blocs indiqués sont acceptés). Pour y arriver, des trous additionnels sont forés pour couvrir les blocs non couverts par le forage d'exploration. Cette étape est appelée le forage de définition.

L'objectif du forage de définition est de minimiser l'incertitude sur la densité de minerai dans les blocs afin d'obtenir un modèle de blocs précis qui peut être utilisé pour étudier la faisabilité de la mine (analyse économique) et planifier l'exploitation minière. Contrairement aux trous d'exploration, les trous de définition doivent être planifiés d'une façon précise et chaque trou couvre un sous-ensemble bien défini de blocs.

Puisque les trous de forage coûtent très cher, les géologues tentent de positionner leurs trous de définition d'une façon qui minimise la quantité de forage (les coûts) tout en assurant la couverture des blocs pertinents. En plus des coûts associés au forage, le déplacement de la foreuse d'un point de collet (position où la foreuse peut être placée pour forer un trou) à un autre coûte cher aussi. Le problème de positionnement des trous de forage consiste donc à choisir les positions des points de collet et les positions, orientations et longueurs des trous qui minimisent le coût total du forage de définition.

Les techniques courantes utilisées pour optimiser le positionnement des trous dans les mines sont peu sophistiquées et ne trouvent généralement pas les positions optimales des trous. Parmi les limitations des approches existantes :

1. Le problème est fréquemment traité en 2D au lieu de 3D.
2. Les algorithmes d'optimisation proposés sont peu efficaces et ne trouvent généralement pas la solution optimale du problème posé.
3. Les coûts de positionnement de la foreuse ne sont pas pris en compte.

De plus, certaines approches existantes posent des hypothèses non réalistes qui compromettent l'optimalité des solutions obtenues ; parmi ces hypothèses :

1. Le nombre de trous est connu à l'avance.
2. La longueur des trous est connue à l'avance.
3. Les trous sont supposés parallèles.
4. L'azimut des trous est connu à l'avance.

Les techniques courantes et leurs limitations sont discutées au chapitre 2.

1.2 Objectif général

L'objectif général de ce projet est de développer une nouvelle approche pour résoudre le problème de positionnement des trous de forage dans les mines qui élimine les limitations des approches existantes et qui permet de réduire les coûts associés au forage de définition.

1.3 Notre Approche

Pour atteindre l'objectif du projet, le problème de positionnement des trous de forage (de définition) dans les mines est modélisé et résolu sous forme d'un problème de recouvrement partiel d'ensembles. Le problème de recouvrement d'ensembles, ou "*Set Covering Problem*" (SCP) en anglais, est un problème très populaire en recherche opérationnelle et a été prouvé *NP-difficile* [42]. Ce problème a été utilisé pour modéliser plusieurs types d'applications industrielles, dont l'assignation d'équipages, la planification de tâches, la localisation d'entrepôts, la construction de circuits imprimés et autres (voir [6] et [25]). Le problème de recouvrement d'ensembles (SCP) peut être défini comme suit. Soit $E = \{e_1, ..., e_n\}$ un ensemble de n éléments et $S = \{s_1, ..., s_m\}$ un univers de sous-ensembles $s_j \subseteq E$. À chaque sous-ensemble s_j est associé un coût c_j. Le SCP consiste à trouver une collection de sous-ensembles $X \subseteq S$ qui couvre tous les éléments de E à un coût minimal.

Le problème de positionnement de trous de forage (PTF) peut être vu comme un SCP où les trous sont représentés par les sous-ensembles du SCP et les blocs sont représentés par les éléments du SCP.

En utilisant le modèle de blocs et les points de collet disponibles, il est possible de générer l'ensemble de tous les trous (appelé l'univers des trous) qui peuvent être forés (qui sont

techniquement réalisables par la foreuse) dans le modèle de blocs 3D. L'univers des trous est fini parce que la foreuse possède des précisions finies en orientation et en longueur. Chaque trou de l'univers a un coût et couvre au moins un bloc. Un bloc est considéré couvert par un trou si la distance orthogonale du centre du bloc au trou est inférieure au rayon de couverture (qui est un paramètre). La figure 1.1 montre un exemple d'univers de trous dans un modèle 2D de blocs où seulement deux points de collet sont montrés à des fins de clarté. Pour illustrer la notion de couverture d'un trou, les blocs couverts par un des trous (le trou en gras) sont colorés en gris. La génération de l'univers des trous est présentée à la section 3.1.

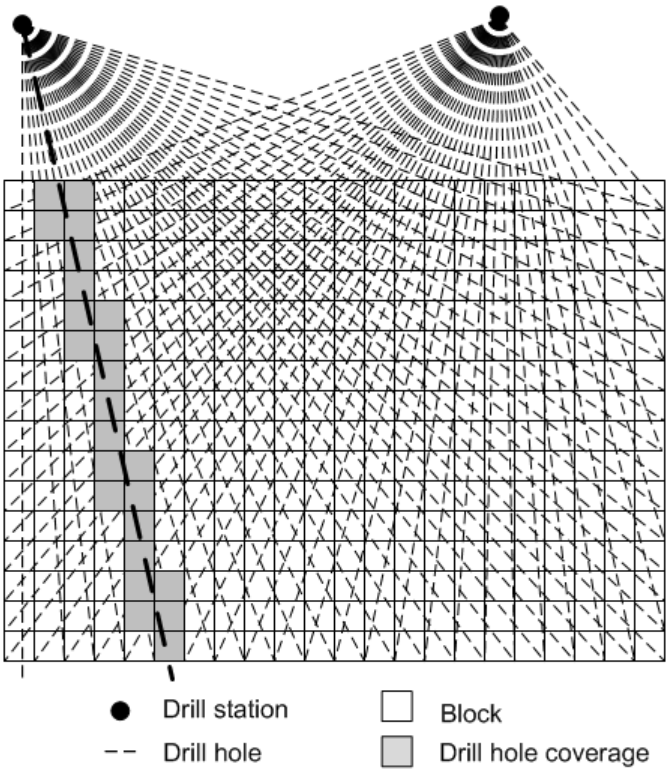

Figure 1.1 Univers de trous 2D

En modélisant le problème PTF comme un SCP, la solution optimale est représentée par une collection de trous qui couvrent tous les blocs ciblés à un coût minimal.

Pour aborder certains aspects du problème PTF qui ne sont pas couverts par le SCP classique, deux nouvelles variantes du SCP partiel sont introduites dans cette thèse (voir chapitre 3). La première variante permet de prendre en compte les coûts de repositionnement de la foreuse et d'éviter les cas extrêmes où le coût de forage investi pour couvrir certains blocs est très élevé par rapport à la valeur qu'apportent ces blocs au forage de définition. La deuxième variante est identique à la première, mais incorpore une contrainte de budget qui permet d'interdire les solutions dont le coût dépasse le budget alloué au forage de définition.

Pour résoudre les modèles proposés (incluant le SCP), plusieurs algorithmes heuristiques sont proposés (voir chapitre 3). Ces algorithmes sont hybrides et basés sur la recherche taboue, la recherche locale itérée et l'algorithme génétique.

Pour valider notre approche, nos modèles et algorithmes sont testés sur plusieurs cas tests réels provenant de l'industrie minière où des réductions des coûts allant jusqu'à 35% ont été réalisées (voir chapitre 9).

1.4 Question de recherche

Est-il possible (et bénéfique) de résoudre le problème de positionnement des trous de forage sous forme d'une variante du problème de recouvrement d'ensembles ?

1.5 Objectifs spécifiques

Les objectifs spécifiques qui nous permettront de répondre à la question de recherche et d'atteindre l'objectif général du projet sont les suivants :

1. Proposer un nouveau modèle d'optimisation qui permet de résoudre le problème de positionnement des trous de forage en 3D. En plus de la position des trous, ce modèle doit permettre l'optimisation du nombre de trous, leurs longueurs, leurs inclinaisons et leurs azimuts. Les coûts de déplacement de la foreuse doivent aussi être pris en compte.
2. Formuler le problème d'optimisation sous forme d'un modèle de programmation mathématique. Le modèle sera basé sur le problème de recouvrement d'ensembles et doit prendre en compte les contraintes liées au positionnement des trous de forage.
3. Développer un algorithme métaheuristique efficace, capable de résoudre le modèle proposé d'une façon optimale ou quasi optimale dans un temps raisonnable.
4. Développer un algorithme métaheuristique efficace pour le problème de recouvrement d'ensembles classique qui trouve des meilleurs résultats que les algorithmes métaheuristiques existants.
5. Démontrer l'efficacité des algorithmes développés en comparant les solutions trouvées par ces derniers aux solutions trouvées par un solveur de programmation mathématique (comme CPLEX) et par d'autres algorithmes existants.
6. Démontrer la pertinence de l'approche proposée sur des cas tests réels du problème PTF en comparant les solutions trouvées par cette approche aux solutions trouvées par les approches utilisées en industrie.

1.6 Hypothèses

Dans cette thèse, nous posons plusieurs hypothèses qui nous permettent de centrer nos efforts sur le volet optimisation du problème PTF, sans s'attarder sur certains aspects géologiques et miniers du problème, qui méritent d'être traités séparément dans une thèse en géologie ou en géostatistique.

Premièrement, nous supposons que les trous de forage ont une trajectoire en ligne droite malgré qu'en pratique, les trous de forage ont tendance à courber. De plus, la courbure d'un trou est généralement imprévisible. D'un autre côté, puisqu'un trou est représenté par un sous-ensemble de blocs, remplacer les lignes droites par des courbes ne nécessite pas de changements dans le modèle et l'algorithme d'optimisation. Cette constatation justifie cette hypothèse et fait en sorte que la courbure des trous peut être abordée séparemment.

La deuxième hypothèse concerne le modèle de couverture binaire utilisé. En fait, nous supposons qu'un bloc est soit complètement couvert par un trou, soit non-couvert par ce dernier (couverture binaire). En pratique, un trou peut couvrir partiellement un bloc et plusieurs trous peuvent collaborer ensemble pour couvrir un bloc. Cette hypothèse est raisonnable parce que dans le forage de définition, le nombre de trous forés est élevé et les trous sont forés en sorte que chaque bloc est couvert de près par un trou.

CHAPITRE 2

REVUE DE LITTÉRATURE

Dans cette section, les techniques courantes utilisées pour optimiser le positionnement des trous de forage dans les mines seront revues. Les lacunes de ces méthodes seront identifiées pour mettre en évidence les besoins traités dans ce projet. Puisque le problème de positionnement des trous de forage sera formulé sous forme d'un problème de recouvrement d'ensembles, nous passerons en revue les variantes du problème de recouvrement d'ensembles afin d'identifier la variante la plus adéquate pour notre problème ou justifier le besoin de créer une nouvelle variante.

2.1 Positionnement des trous de forage

Le problème de positionnement des trous de forage dans l'exploration minérale a été peu traité dans la littérature. La majorité des algorithmes proposés dans la littérature traitent le problème en 2D au lieu de 3D ; ce qui ne permet pas toujours d'atteindre la solution optimale. De plus, ces algorithmes sont fréquemment développés par des géostatisticiens qui s'intéressent principalement au calcul de l'incertitude sur les blocs. Les techniques d'optimisation utilisées dans ces algorithmes sont généralement peu efficaces.

Saikia et Sarkar présentent un algorithme pour l'optimisation des trous de forage dans une mine de charbon à Jharia [82]. Les auteurs supposent que tous les trous sont parallèles et qu'ils sont placés dans une grille 2D uniforme (les espacements verticaux et horizontaux entre les trous voisins sont égaux). L'algorithme proposé optimise l'espacement entre les trous. L'algorithme démarre avec un espacement initial qui est suffisamment large. À chaque itération, l'espacement entre les trous est réduit et la variance de krigeage correspondante est calculée (le Krigeage [91] est une méthode utilisée pour calculer l'incertitude sur les blocs). La réduction de l'espacement entre les trous implique une hausse des coûts de forage puisque le nombre de trous à forer augmente. D'un autre côté, l'augmentation du nombre de trous permet de réduire l'incertitude sur les blocs. L'algorithme s'arrête lorsque la réduction de la variance de krigeage causée par une réduction d'espacement n'est pas suffisamment élevée pour justifier l'augmentation des coûts.

Pan propose une approche géostatistique complète qui permet de classifier les blocs de minéralisation dans le modèle de blocs en assignant à chaque bloc une valeur qui reflète sa priorité par rapport au forage de définition [76]. L'auteur souligne le fait que le forage de

définition est lui-même séparable en trois étapes. La première étape permet de vérifier la continuité de minéralisation. Dans la seconde étape, en se basant sur la continuité de minéralisation, des trous sont forés à l'extérieur de la zone de minéralisation connue afin d'étendre le modèle de blocs. Dans la dernière phase, qui est la phase la plus coûteuse, les blocs de hautes densités de minéralisation sont visés par les trous. C'est d'ailleurs cette troisième phase que nous traitons dans ce projet. L'auteur commence par présenter une description détaillée du problème de forage de définition. Par la suite, une nouvelle méthode de classification des blocs est présentée. Pour assigner une priorité à chaque bloc, trois facteurs sont pris en compte : attributs géologiques, indications de minéralisation et la teneur en minerai. En se basant sur ces facteurs, les blocs sont classifiés en cinq catégories : 1) "Probable reserves" sont les blocs de hautes densités dont la présence a été confirmée par les trous d'exploration. 2) "Indicates reserves" sont les blocs qui sont potentiellement riches en minerai d'une valeur économique très élevée, mais dont le niveau d'incertitude n'est pas tout à fait satisfaisant. Ces blocs n'ont pas été atteints par les trous d'exploration. 3) "Indicated resources" contient des blocs ayant des propriétés similaires à ceux dans "indicated reserves" mais dont la valeur économique est moins élevée. 4) "Possible resources" représente les blocs à haute incertitude qui contiennent possiblement une quantité significative de minerai. 5) "Barren" contient les blocs qui sont hors de la zone géologique favorable et qui ne sont pas visés par le forage de définition. "Indicated reserves", "Indicated resources" et "Possible resources" sont les blocs visés par le forage de définition où les blocs de la catégorie "Indicated reserves" ont la priorité la plus élevée, les blocs de la catégorie "Indicated ressources" ont le deuxième niveau de priorité et les blocs "Possible resources" ont le troisième niveau de priorité. Suite à la classification des blocs, les trous sont positionnés manuellement par un expert dans le domaine. Tous les trous sont supposés parallèles et seulement l'espace 2D est considéré.

Deux articles, publiés récemment (2011), présentent un premier pas pour résoudre le problème de positionnement des trous de forage en trois dimensions [86, 85].

Soltani, Hezarkhani, Tercan et Karimi proposent un algorithme génétique pour optimiser le forage de définition en 3D [86]. En plus de la position des points de départ des trous, la longueur des trous est aussi optimisée. Un chromosome (représentation d'une solution dans l'algorithme génétique) contient les coordonnées et la longueur de M trous $[x_1, y_1, l_1, x_2, y_2, l_2, ..., x_M, y_M, l_M]$ où M est un paramètre choisi au départ. L'absence de l'orientation des trous et de la coordonnée z des points de départ dans le chromosome suggère que les trous sont supposés parallèles et que les points de départ des trous sont considérés dans le même plan. De plus, le fait que le nombre de trous à forer (M) est choisi au départ peut compromettre l'optimalité de la solution trouvée. Les auteurs présentent les grandes lignes de l'algorithme génétique, mais ne décrivent pas les opérateurs génétiques utilisés (croisement,

mutation, sélection, etc).

Soltani est Hezarkhani, qui sont d'ailleurs coauteurs dans l'article précédent, proposent une approche similaire à la précédente, mais où l'inclinaison ("dip") des trous représente la troisième dimension à optimiser [85]. La différence principale avec leur première approche est que l'inclinaison des trous est optimisée au lieu de la longueur, mais pas les deux. De plus, la métaheuristique recuit simulé (SA) est utilisée à la place de l'algorithme génétique. La longueur, le nombre et l'azimut des trous sont supposés constants. Cette approche souffre des mêmes limitations que la précédente.

2.1.1 Limitations des méthodes existantes

Les méthodes existantes de positionnement des trous de forage contiennent deux modules principaux : un module géostatistique qui permet de classifier les blocs sous différentes catégories et un module d'optimisation qui permet d'optimiser le positionnement des trous en se basant sur la classification des blocs. Certaines méthodes de classification traitent uniquement l'incertitude sur les blocs et d'autres traitent des facteurs additionnels comme les attributs géologiques, la teneur en minerai, etc. Dans cette thèse, nous nous intéressons principalement aux limitations des méthodes existantes au niveau du module d'optimisation. Ces limitations peuvent être résumées comme suit :

1. Le problème est fréquemment traité en 2D au lieu de 3D.
2. Les algorithmes d'optimisation proposés sont peu efficaces et ne trouvent généralement pas la solution optimale au problème posé.
3. Les coûts de positionnement de la foreuse ne sont pas pris en compte.
4. La contrainte de budget n'est pas prise en compte.
5. Dans le cas 3D, les hypothèses suggérées compromettent l'optimalité des solutions. Parmi les hypothèses rencontrées :
   - Le nombre de trous est connu à l'avance.
   - La longueur des trous est supposée constante.
   - Les trous sont supposés parallèles.
   - L'azimut des trous est connu à l'avance.

Notre but est d'éliminer, dans la mesure du possible, les limitations du module d'optimisation en modélisant le problème de positionnement des trous de forage sous forme d'un problème de recouvrement d'ensembles.

2.2 Recouvrement d'ensembles (SCP)

Il existe plusieurs variantes du SCP. Lorsque tous les éléments doivent être couverts exactement une fois, le problème est appelé *partitionnement d'ensembles* [8]. Lorsque tous les éléments doivent être couverts plus qu'une fois, on parle de *recouvrement multiple d'ensembles* [47]. Lorsqu'il n'est pas nécessaire (ou possible) de couvrir tous les éléments de E pour répondre à d'autres critères ou contraintes, on parle de la famille des problèmes de recouvrement partiel d'ensembles [62, 57, 94, 41, 10]. Le recouvrement partiel d'ensembles (PSCP) est une généralisation du SCP. Plusieurs variantes du PSCP ont été étudiées dans la littérature : la variante appelée *k-set covering* [4] consiste à couvrir au moins k éléments à un coût minimal. Une autre variante provenant des problèmes de localisation d'entrepôts [94], où le nombre maximal d'entrepôts (budget maximal) permis est limité, a aussi été étudiée. À cause du budget limité, certains éléments ne peuvent être couverts. Un cas particulier du problème de localisation d'entrepôts est appelé *the budgeted maximum coverage location problem*. Dans ce problème, un gain g_i est attribué à chaque élément e_i et le but est de choisir une collection de sous-ensembles qui maximise le gain total ($\sum g_i$ couverts) tout en respectant la contrainte de budget. Une nouvelle variante appelée *prize-collecting set cover problem* a été proposée dans [57]. Dans cette variante, un profit p_j est attribué à chaque élément $e_i \in E$. L'objectif est de trouver une sous-collection $X \subseteq S$ de coût minimal, et tel que le profit total des éléments couverts par X soit supérieur ou égal à un profit minimum spécifié.

Dans le problème de positionnement des trous de forage dans les mines, à cause du budget limité attribué au forage de définition, il est parfois impossible de couvrir tous les blocs. Pour cette raison, le problème PTF ne peut être modélisé à l'aide du SCP classique mais plutôt comme une variante du recouvrement partiel d'ensembles. La variante du PSCP qui semble la plus adéquate pour modéliser le problème PTF est celle de localisation d'entrepôts avec budget (*the budgeted maximum coverage location problem*). Le gain attribué à un bloc est proportionnel au coût qu'on est prêt à débourser pour couvrir ce bloc. Cette valeur dépend de la valeur économique du bloc et de l'incertitude sur cette valeur. Les coûts des sous-ensembles sont proportionnels à la longueur des trous. L'objectif est de maximiser la somme des gains des blocs couverts tout en respectant la contrainte de budget. Ce modèle n'est pas tout à fait adéquat pour deux raisons : 1) les coûts de déplacement de la foreuse ne sont pas pris en compte 2) la solution optimale peut contenir des trous qui couvrent très peu de blocs. L'exemple qui suit clarifie ce deuxième point. Supposons que :

— Le budget total attribué au forage de définition est de dix millions de dollars.

— La solution optimale contient 27 trous ayant un coût total de 9.5 millions de dollars.

— La solution optimale couvre tous les blocs sauf un bloc dont le gain est égal à cent

dollars.

— Il existe un trou dont le coût est égal à 450 mille dollars qui couvre le bloc non couvert.

Selon le modèle de localisation d'entrepôts avec budget, il est bénéfique d'ajouter le trou de 450 mille dollars à la solution pour couvrir le bloc dont le gain est égal à cent dollars parce que la somme des gains (la valeur de l'objectif) sera augmentée sans violer la contrainte de budget. Dans la vraie vie, forer un trou de 450 mille dollars pour couvrir un tel bloc est hors de considération. En conséquence, le modèle doit être ajusté pour éviter de tels scénarios (appelés des solutions extrêmes) et pour prendre en compte les coûts de déplacement de la foreuse.

CHAPITRE 3

MÉTHODOLOGIE

Tel que mentionné précédemment, notre approche est basée sur la modélisation du problème PTF sous forme d'une variante du SCP. Trois variantes du SCP ont été étudiées dans cette thèse. Chaque variante est utilisée pour mieux couvrir certains aspects et contraintes du problème PTF.

Le premier modèle est une variante du recouvrement partiel d'ensembles impliquant la maximisation de profit (PMSCP pour *Profit Maximization Set Covering Problem*). Cette variante permet de prendre en compte les coûts de repositionnement de la foreuse en introduisant la notion de groupe de sous-ensembles. En fait, tous les trous forés à partir d'un même point de collet appartiennent au même groupe auquel est associé un coût fixe. De plus, l'objectif de maximisation de profit permet d'éviter les solutions extrêmes où le coût de forage investi pour couvrir certains blocs est très élevé par rapport à la valeur qu'apportent ces blocs au forage de définition (par exemple, une solution est considérée extrême si un long trou est utilisé pour couvrir un seul bloc). Pour résoudre le PMSCP, un algorithme métaheuristique hybride combinant la recherche locale itérée et la recherche taboue (ITS pour *Iterated-Tabu-Search*) a été développé. Pour évaluer l'efficacité de cet algorithme, nous comparons les solutions trouvées avec ce dernier aux solutions optimales ou quasi-optimales trouvées à l'aide du solveur de programmation mathématique CPLEX ; nous allouns à CPLEX le temps et les ressources nécessaires pour trouver des solutions optimales ou quasi-optimales et les gaps d'optimalité associés à ces solutions. Il est important de noter que nous ne comparons pas ITS à CPLEX mais utilisons CPLEX pour obtenir des valeurs références qui nous permettent d'évaluer la qualité des solutions obtenues par ITS. Au lieu, nous comparons ITS à un algorithme mémétique pour le SCP qui a été adapté pour le SCP. Cet algorithme a été choisi parce qu'il est un des algorithmes les plus populaires pour le SCP (qui est un problème similaire) et est facilement adaptable pour résoudre le PMSCP. Le modèle PMSCP et l'algorithme ITS sont discutés en détail dans l'article présenté au chapitre 4 qui a été publié dans la revue *Journal of Heuristics* [20]. L'analyse expérimentale présentée dans l'article montre que l'algorithme développé est très efficace.

La deuxième variante étudiée est le SCP classique. En fait, il arrive fréquemment des cas où il est requis de couvrir tous les blocs et où les positions de la foreuse sont connues à l'avance. De tels cas arrivent à la fin du forage de définition lorsque les informations sur les blocs sont suffisantes pour identifier les blocs qui doivent être couverts. Bien que ces cas peuvent

être modélisés avec la première variante (puisque la première variante est une généralisation du SCP), ces cas peuvent être modélisés et résolus plus efficacement sous forme d'un SCP. Puisque tous les blocs doivent être couverts, le problème consiste à choisir l'ensemble des trous (sous-ensembles) qui couvrent tous les blocs (éléments) à un coût minimal.

Pour résoudre le SCP, un algorithme métaheuristique hybride combinant l'algorithme génétique et la recherche taboue a été développé. Cet algorithme est basé sur une nouvelle formulation de programmation mathématique du SCP que nous avons développée pour faciliter la résolution du SCP à l'aide de métaheuristiques. En fait, la formulation classique du SCP est difficile à résoudre en utilisant des métaheuristiques parce que la fonction objectif (qui représente une minimisation du coût) et la contrainte de couverture totale (qui impose que tous les éléments doivent être couverts) guident la recherche dans deux directions opposées. Pour remédier à cette difficulté, la contrainte de couverture est remplacée par des pénalités dans la fonction objectif de la nouvelle formulation. Bien que des approches de pénalités similaires existent dans la littérature, la différence principale entre notre approche et les approches de pénalités courantes est au niveau du choix des pénalités. En fait, nous choisissons les plus petites pénalités possibles qui permettent d'assurer la faisabilité des solutions sans perturber la recherche ; tandis que les approches de pénalités existantes utilisent soit des pénalités élevées qui perturbent la recherche, soit des basses pénalités qui ne garantissent pas la faisabilité des solutions, soit des pénalités dynamiques qui sont difficiles à calibrer. La comparaison de notre approche aux approches de pénalités courantes ainsi que la nouvelle formulation sont décrites dans l'article présenté au chapitre 5 qui a été publié dans la revue *IRSN Operations Research* [19]. L'algorithme métaheuristique développé pour résoudre le SCP est décrit dans l'article présenté au chapitre 6 qui a été soumis à la revue *Journal of Mathematical Modelling and Algorithms in Operations Research.*

Comme toutes variantes du recouvrement d'ensembles, les modèles proposés supposent l'existence d'un univers de sous-ensembles à partir duquel les solutions seront choisies. Dans le cas du problème PTF, l'univers des sous-ensembles est l'univers de tous les trous possibles. Les techniques de génération de l'univers des trous sont discutées à la section suivante (Section 3.1).

Les modèles proposés ont été conçus pour être flexibles et adaptables à une grande variété de situations qu'un géologue peut rencontrer. L'utilisation et l'adaptation des modèles ainsi que plusieurs scénarios typiques d'utilisation sont discutés aux sections 8.1 et 8.2.

3.1 Génération de l'univers des trous

Dans toutes les variantes du problème de recouvrement d'ensembles, une collection de sous-ensembles est choisie pour maximiser ou minimiser un certain objectif. Ceci requiert l'existence d'un univers de sous-ensembles à partir duquel la collection de sous-ensembles sera choisie. Dans le contexte du positionnement des trous de forage, cet univers de sous-ensembles est l'ensemble de tous les trous possibles qui peuvent être forés dans la mine. Il est donc important de développer un algorithme permettant de générer tous les trous possibles et de convertir les trous en sous-ensembles de blocs où un coût est attribué à chaque sous-ensemble. Les trous générés doivent être réalisables en pratique. En d'autres mots, ces trous doivent respecter les contraintes liées au forage et à la structure géologique du terrain de minéralisation.

En pratique, la trajectoire d'un trou n'est pas toujours prévisible. Bien que les trous de forage soient habituellement planifiés comme des lignes droites, les trous ont tendance à courber et la vraie trajectoire est très difficile à prévoir. Dans cette thèse, nous supposons que la trajectoire d'un trou de forage est une ligne droite. Cette hypothèse est d'ailleurs raisonnable dans la plupart des cas et est utilisée actuellement par la majorité des géologues. En fait, la prédiction de la trajectoire des trous de forage est un sujet très complexe et mérite d'être étudiée séparément dans une thèse en géologie ou en géostatistique.

En supposant que la trajectoire d'un trou de forage est une ligne droite, l'approche que nous utilisons pour générer l'univers des trous est la suivante. À partir de chaque point de collet, nous générons tous les trous qui intersectent le modèle de blocs. Géométriquement, le nombre de trous de forage qui peuvent être générés à partir d'une position donnée de la foreuse est infini. En pratique, la précision de la foreuse est finie et par conséquent le nombre de trous possibles est fini. Tel que discuté à la section 2, un trou est caractérisé par son azimut, dip, longueur et son point de collet (qui est identique à la position de la foreuse). En considérant toutes les valeurs possibles d'azimut, dip et longueur qui peuvent être réalisées par la foreuse (selon la précision de la foreuse), tous les trous possibles peuvent être générés. Seulement les trous qui couvrent au moins un bloc sont ajoutés à l'univers des trous.

L'algorithme de génération des trous est présenté dans l'algorithme 1. La fonction *getEndPoint(collarPoint, L, Dip, Az)* détermine la position de l'extrémité finale du trou de forage à partir des paramètres *collarPoint* (point de départ), L (longueur), Dip et Az (azimut). La fonction *getCoveredBlocks(collarPoint, endPoint)* détermine les blocs qui sont couverts par le trou défini par les extrémités *collarPoint* et *endPoint*. Un bloc est couvert par un trou donné si la distance orthogonale du centre du bloc au trou est inférieure ou égale au rayon de couverture utilisé (qui est un paramètre). La figure 3.1 montre un exemple d'univers de

trous généré à l'aide de l'algorithme 1.

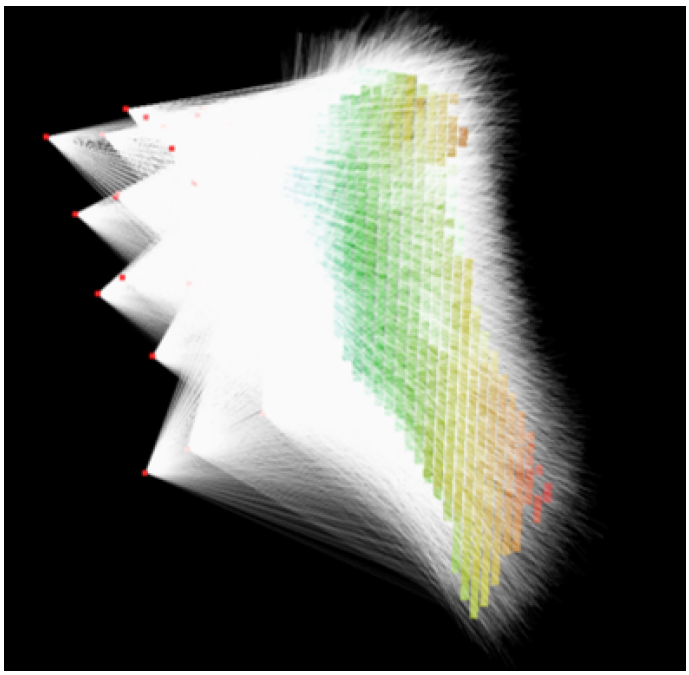

Figure 3.1 Univers des trous

Le coût des trous est généralement proportionnel à leur longueur mais la distribution de coût peut être ajustée selon l'objectif à atteindre. Par exemple, si les trous courts sont préférés aux trous longs, il est possible d'utiliser une fonction de coût qui augmente d'une façon quadratique par rapport à la longueur ($C(L) = L^2$).

Dans le reste de cette thèse, nous utilisons la technique de génération de trous présentée dans cette section pour créer l'univers des trous.

Algorithm 1 Generate all candidate drill holes

```
drillholes = {}

for all collar points do
  L_min ← minimum drill hole length from the current collar point.
  L_max ← maximum drill hole length from the current collar point.
  Dip_min ← minimum dip angle from the current collar point.
  Dip_max ← maximum dip angle from the current collar point.
  Az_min ← minimum azimuth angle from the current collar point.
  Az_max ← maximum azimuth angle from the current collar point.

  L ← L_min
  while (L ≤ L_max) do
    Dip ← Dip_min
    while (Dip ≤ Dip_max) do
      Az ← Az_min
      while (Az ≤ Az_max) do
        endPoint ← getEndPoint(collarPoint, L, Dip, Az)
        blocks ← getCoveredBlocks(collarPoint, endPoint)
        if (size(blocks) > 0) then
          if (drillholes does not contain a cheaper set covering the same blocks) then
            drillholes.addSet(collarPoint,endPoint,blocks)
          end if
        end if
        Az ← Az + Az_step
      end while
      Dip ← Dip + Dip_step
    end while
    L ← L + L_step
  end while
end for
```

CHAPITRE 4

ARTICLE 1 : AN ITERATED-TABU-SEARCH HEURISTIC FOR A VARIANT OF THE PARTIAL SET COVERING PROBLEM

Auteurs : Nehme Bilal, Philippe Galinier et Francois Guibault
Revue : *Journal of Heuristics*
Référence : [20]

Abstract

In this paper, we propose a heuristic algorithm to solve a new variant of the partial set covering problem. In this variant, each element e_i has a gain g_i (i.e., a positive profit), each set s_j has a cost c_j (i.e., a negative profit), and each set s_j is part of a unique group G_k that has a fixed cost f_k (i.e., a negative profit). The objective is to maximize profit and it is not necessary to cover all of the elements. We present an industrial application of the model and propose a hybrid heuristic algorithm to solve it ; the proposed algorithm is an iterated-local-search algorithm that uses two levels of perturbations and a tabu-search heuristic. Whereas the first level of perturbation diversifies the search around the current local optimum, the second level of perturbation performs long jumps in the search space to help escape from local optima with large basins of attraction. The proposed algorithm is evaluated on thirty real-world problems and compared to a memetic algorithm. Computational results show that most of the solutions found by ITS are either optimal or very close to optimality.

4.1 Introduction

The set covering problem (SCP) is an *NP-hard* optimization problem [42] that has been extensively studied in operations research and combinatorial optimization. The SCP can be defined as follows : let $E = \{e_1, ..., e_m\}$ be a universe of elements and $S = \{s_1, ..., s_n\}$ be a collection of subsets $s_j \subset E$, where $\bigcup s_j = E$, with $j = 1 \ldots n$. Each set s_j covers at least one element of E and has a cost $c_j > 0$. The objective is to find a sub-collection of sets $X \subseteq S$ that covers all of the elements in E at a minimal cost. The SCP has been applied to a wide range of industrial applications including scheduling, manufacturing, service planning and location problems [22, 61, 6].

To address the specific needs of some applications, the partial set covering problem (PSCP) has been introduced. The PSCP is a generalization of the SCP where it is either

not necessary or not possible to cover all of the elements of E because of other objectives or constraints. Several variants of the PSCP have been proposed in the literature. In the *k-set covering* variant [4], the aim is to cover at least k elements at a minimal cost. In the facility location variant [94], the total number of facilities (i.e., total budget) allowed to cover the demand points (i.e., the elements) is usually limited, and as a result, some demand points cannot be covered. The *budgeted maximum coverage location problem* [54] is a particular case of facility location problems and it involves choosing a subset of sets from a collection of sets of weighted elements, to maximize the total weight that is covered under a given budget constraint. Another variant of the PSCP called the *prize-collecting set cover problem* has been proposed in [57]. In this variant, a profit p_j is associated with each element $e_i \in E$. The objective is to find a minimum cost subset of S such that the total profit of the covered elements is greater than or equal to a specified profit bound.

In order to solve an industrial problem (which is described in Section 4.2.1), we introduce a profit-maximization variant of the partial set covering problem (PMSCP). In this variant, each element e_i has a gain g_i (i.e., a positive profit), each set s_j has a cost c_j (i.e., a negative profit), and each set s_j is part of a unique group G_k that has a fixed cost f_k (i.e., a negative profit). The fixed cost of a group is paid only once, namely, whenever a set of the group is added to the solution. The objective is to maximize the total profit represented by the total gain of the covered elements minus the total cost of the sets and groups that are used in the solution. The profit-maximization objective encapsulates the idea that when the additional cost that is required to cover certain uncovered elements is higher than the additional revenue ($\sum g_i$) associated with these elements, it is better to leave these elements out of the cover.

To solve the PMSCP, we propose an iterated-tabu-search algorithm (ITS) that is an iterated-local-search (ILS) algorithm, where a tabu-search heuristic (TS) is used for local-search. An important characteristic of our algorithm is the use of a second perturbation operator in the ILS framework that helps escaping from local optima and hence, increases the chances of reaching optimal solutions.

Because the PMSCP is a new problem that has not previously been solved in the literature, there are no existing algorithms to which we can compare our heuristic. Instead, we adapted one of the best heuristics that has been developed for the SCP (which is a very similar problem) and compared our heuristic to it. The adapted algorithm has been developed by [16] and is described in Section 4.4. The two algorithms are compared on 30 real world test problems (Section 4.5). Additionally, limited experiments using the general purpose solver CPLEX are performed to validate our results.

The main contributions of this work are the use of tabu-search and a second perturbation operator in the ILS framework. To our knowledge, iterated-local-search has not previously

been used to solve the partial or standard set covering problem. In addition, we present a new variant of the PSCP and a new practical application of it.

4.2 Modeling a mining-industry application with the PMSCP

4.2.1 The mining application

We originally developed our profit-maximization variant of the partial set covering problem to solve a mining-industry problem called *drillholes placement*. The first steps in the mining cycle are exploration and feasibility. In the exploration stage, geologists start by estimating the potential locations of mineral deposits. Then, they drill many long holes inside the mine and extract samples from these potential sites for analysis and to confirm or adjust their estimations. Because drilling is expensive, geologists and mining engineers try to position their holes to cover most potential sites with a minimum amount of drilling. Minimizing the amount of drilling involves choosing the number, location, orientation and length of each drill hole to minimize the total cost of drilling. In addition to the cost of drilling, moving the drill-platform from one location to another is also expensive. The locations where the drill-platform can be positioned are called *drill stations*.

Usually, geologists express their estimations in the form of a 3-D model ; in the mining industry, this model is often called a *block model*. A block model is a set of cubic blocks of equal size where each block is characterized by the estimated grade of the minerals it may contain. Using this block model and the available drill stations, it is possible to create a set that contains all of the potential drill holes that can be drilled inside the mine. In this set of drill holes, each drill hole can have any length and orientation that is feasible using the drill, which can be rotated both horizontally and vertically. In addition, each drill hole covers at least one block and the drill holes are grouped by drill stations. A block is said to be covered by a given drill hole if the orthogonal distance from the center of the block to the drill hole is smaller than a given value. Figure 4.1 shows an example of a set of potential drill holes in a 2-D block model where only two drill stations are shown for illustration purposes. To illustrate the drill hole coverage, the blocks covered by one of the drill hole (highlighted in bold) are shown in gray.

Each drill hole has a cost that is proportional to its length and each block has a gain that is proportional to the importance of covering that block. Because the blocks are only explored (not extracted) in the exploration stage, the gain attributed to each block is the highest cost that we would find it worthwhile to pay to cover that block (not the value of the minerals that are inside that block). A geostatistical classification method can be used to separate important blocks from less important ones, which makes the attribution of the

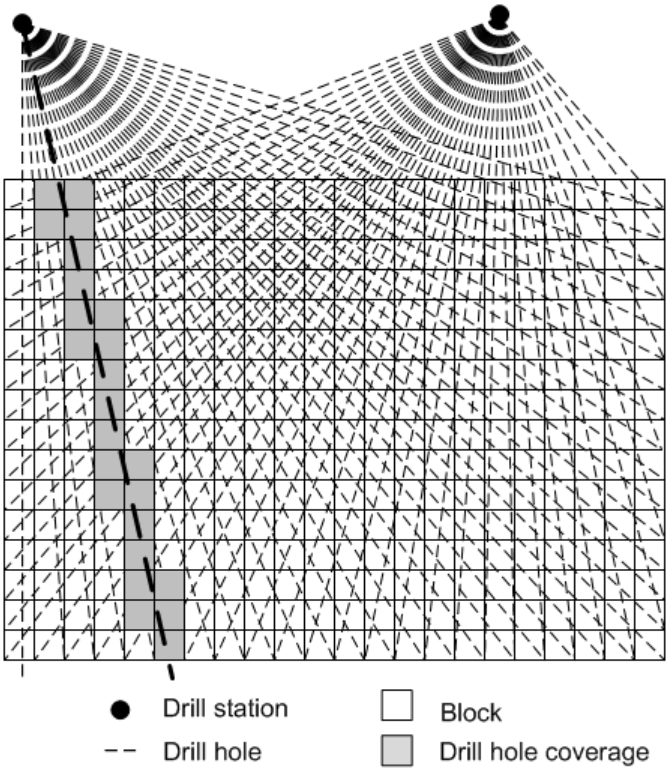

Figure 4.1 Set of potential drill holes

gains easier. For instance, the classification method proposed by [76] allows separating the blocks into five different categories.

From the set of all possible drill holes, we want to choose a subset of drill holes that maximizes the profit represented by $\sum$gains (blocks covered) $-$ $\sum$costs (drill holes selected) $-$ $\sum$costs (drill relocation).

In order to solve this problem with our PMSCP model, we need to generalize the concepts of drill holes, blocks and drill stations. Because a drill hole covers a set of blocks and because the drill holes are grouped by drill stations, a block can be modeled as an element (e_i), a drill hole can be modeled as a set of elements (s_j) and a drill station can be modeled as a group of sets (G_k). Moreover, each element has a gain (g_i), each set has a cost (c_j) and each group has a fixed cost (f_k). Because the cost of moving the drill to a new location is paid exactly once, the cost of a group f_k is modeled as a fixed cost.

4.2.2 The PMSCP model

The PMSCP is a maximization problem that can be formulated as follows. Let

— $A^{m\times n}$ be a zero-one matrix where $a_{ij} = 1$ if the element e_i is covered by the set j, and $a_{ij} = 0$ otherwise.

— $B^{l\times n}$ be a zero-one matrix where $b_{kj} = 1$ if the set s_j is part of the group G_k, and $b_{kj} = 0$ otherwise.

— $X = \{x_1, x_2, ..., x_n\}$ where $x_j = 1$ if the set s_j (with cost $c_j > 0$) is part of the solution, and $x_j = 0$ otherwise.

— $Y = \{y_1, y_2, ..., y_m\}$ where $y_i = 1$ if the element e_i (with a gain $g_i > 0$) is covered in the solution, and $y_i = 0$ otherwise.

— $GR = \{gr_1, gr_2, ..., gr_l\}$ where $gr_k = 1$ if at least one set of the group G_k (with a cost $f_k > 0$) is part of the solution, and $gr_k = 0$ otherwise.

Maximize

$$\sum_{i=1}^{m} g_i y_i - \sum_{j=1}^{n} c_j x_j - \sum_{k=1}^{l} f_k gr_k \tag{4.1}$$

subject to

$$y_i \leq \sum_{j=1}^{n} a_{ij} x_j, \qquad i = 1, ..., m \tag{4.2}$$

$$b_{kj} x_j \leq gr_k, \quad j = 1, ..., n; k = 1, ..., l \tag{4.3}$$

$$x_j, y_i, gr_k \in \{0, 1\} \tag{4.4}$$

Each set is part of exactly one group and each group must contain at least one set. Constraint (4.2) implies that an element is covered if at least one of the sets that covers it is part of the solution, and constraint (4.3) implies that a group is part of the solution if at least one of the sets that it contains is part of the solution. In fact, the purpose of constraints (4.2) and (4.3) is to keep track of which elements are covered and which groups are used in a given configuration.

Throughout all this paper, the term *solution* or *configuration* refers to the collection of selected sets (i.e, all of the sets s_j such that $x_j = 1$).

4.3 The proposed ITS algorithm

In this work, we propose a hybrid heuristic algorithm to solve the PMSCP. The proposed algorithm is hybrid because it combines two metaheuristics : iterated-local-search (ILS) and tabu-search (TS).

Iterated-local-search is a simple but very effective metaheuristic that has been successfully applied to many difficult optimization problems, such as the traveling salesman problem [66, 52, 53], scheduling problems [29, 87, 65, 9, 58], graph partitioning [67, 68] and MAX-SAT [11]. The aim of iterated-local-search is to explore the space of local optima in an efficient and effective way. ILS manipulates a single solution using two main operators : a local-search operator and a perturbation operator. In each iteration, a new starting point is created using the perturbation operator and a new local optimum is found using the local-search operator. The new local optimum is kept if it passes a given acceptance criterion ; otherwise, a new iteration is performed using the previous local optimum.

The perturbation and local-search operators must be carefully designed to work well together. Furthermore, the perturbation must be large enough to ensure that it is not negated by the local-search operator and small enough to avoid a random-restart behavior. A complete description and guide for building effective iterated-local-search algorithms is presented in [64].

Tabu-search (TS) is a local-search algorithm that uses a search history to escape from local optima and cycles. The search history of TS is saved in a list, which is called the tabu-list. Whenever a move is performed, the reverse move is added to the tabu-list. A move stays in the tabu-list for a limited number of iterations (equal to the length of the tabu-list). A move is forbidden as long as it is part of the tabu-list. Another important component of tabu-search is the aspiration criterion that allows a tabu move to be performed in some circumstances. A typical aspiration criterion, which is actually used in this paper, allows a tabu move to be performed when the resulting solution is better than the current best solution. A full

description of tabu-search can be found in [44, 46, 95].

To achieve better results, many authors have combined iterated-local-search with other heuristics such as the genetic algorithm [1], variable neighborhood descent (VND) [27], GENIUS [40] and tabu-search (TS) [71, 83, 70, 75]. When tabu-search is used as the local-search operator in ILS, the resulting ILS algorithm is often called *iterated-tabu-search*.

The algorithm proposed in our paper is an iterated-tabu-search (ITS) algorithm with two levels of perturbations : whereas the first level of perturbation diversifies the search around the current local optimum, the second level of perturbation performs long jumps in the search space to help escape from local optima with large basins of attraction. In each iteration, the first perturbation operator is invoked ; then, a short run of tabu-search is performed. The second perturbation operator is invoked if the current best solution is not improved after $stag_{(ITS)}$ iterations, where $stag_{(ITS)}$ is a parameter. The acceptance criterion that is used only allows a better solution to replace the current local optimum at the end of each iteration. The algorithm stops after a limited computation time.

4.3.1 Solution-representation and objective function

We use a binary-string solution representation in ITS where the ith bit is equal to one if set i is part of the solution, and the ith bit is equal to zero otherwise.

Let X be a given configuration, S_X be the sets used in X, E_X be the elements covered by X and G_X be the groups used in X. Then, the objective-score of X can be calculated as follows :

$$\text{score}(X) = \sum_{e \in E_X} \text{gain}(e) - \sum_{s \in S_X} \text{cost}(s) - \sum_{g \in G_X} \text{cost}(g)$$

4.3.2 Perturbation operators

As mentioned earlier, we use two perturbation operators in ITS. The first perturbation operator (see Algorithm 2) diversifies the search around the current best solution by randomly removing sets from (or adding sets to) the current configuration. This operator iterates over the bits of value 1 and flips each bit to 0 with a probability p. Afterward, if n bits have been flipped from 1 to 0, n new bits are randomly chosen and flipped from 0 to 1 (if a bit is already equal to one, it is left at 1). The goal is to ensure that the number of bits that have been flipped from 0 to 1 is less than or equal to the number of bits that have been flipped from 1 to 0 ; the reason is that if too many bits are flipped from 0 to 1, most of the sets will be redundant and the local-search operator will spend excessive time removing redundant sets. This behavior only occurs if the density of ones in a redundant-free solution is small, which is in fact the case for our test problems. The probability p (or the strength of the perturbation

operator) must be low enough to avoid a random-restart behavior but high enough to ensure that the perturbation is not immediately canceled by the local-search operator.

Algorithm 2 perturb$_1$(S)

```
n ← 0
for (i = 1 : numOfSets(S)) do
  if ( flip-coin(p) = true ) then {p is the perturbation strength}
    S ← remove-set(S, i)
    n ← n + 1
  end if
end for
for (i = 1 : n) do
  j = random-int( 1, numOfSets(S) )
  S ← add-set(S, j)
end for
return S ;
```

The analysis of the execution traces that were produced in preliminary testing revealed that ITS have difficulties escaping from certain local optima because of the fixed cost that is associated with the groups (these difficulties have also been confirmed by the experimental results presented in Tables 4.6 and 4.7). In fact, when few groups are added to the solution, their associated fixed costs are paid ; then, ITS tends to keep adding sets to (or removing sets from) the solution using these groups instead of exploring new groups. This happens because the costs of the selected groups have already been paid, and as a result, it is more expensive to pay for other unvisited groups and explore their sets. Therefore, certain regions of the search space, that may in fact contain the global optimum become difficult to reach. To overcome this weakness of ITS, a second perturbation operator that performs long jumps in the search space is used. This operator is invoked when the search is stagnant for a given number of iterations ($stag_{(ITS)}$ in Algorithm 5). The long jumps consist of forcing a group into or out of the solution for a fixed number of iterations (F in Algorithm 3). If the solution is not improved, the group is released and another group is forced (only one group is forced at a time). When a new best solution is found, the forced group is released, the stagnancy counter is reset and the search is continued (see Algorithm 5).

Let M be a very large constant such that $M > \sum(g_i)$. To force a group out of the solution, we remove all of its sets from the solution and increase its fixed cost by M to make it unaffordable. Similarly, to force a group into the solution, we decrease its fixed cost by M to make it very attractive (the cost of the group becomes negative, and therefore, adding any set from the group to the solution increases the objective score). To release a group,

we restore its original cost. During evaluation, if a group was forced into the solution, the original cost of the group must be used instead of the modified cost.

The pseudo-code of the second perturbation operator is presented in Algorithm 3, where F is the number of iterations during which a group is forced into (or out of) the solution.

Algorithm 3 perturb$_2$(S^*, forcedIters, currentGroup, stagnationCounter)

```
if (forcedIters == 0) then
  S* ← force-group(S*, currentGroup);
else if (forcedIters == F) then {if a group was forced for F iterations, release it and
force the next group}
  S* ← release-group(currentGroup);
  currentGroup ← next-group();
  forcedIters ← 0;
  if (is-invalid(currentGroup)) then {if we already tried to force all the groups}
    currentGroup ← first-group();
    stagnationCounter ← 0;
    return S*;
  end if
  S* ← forceGroup(S*, currentGroup);
end if
forcedIters ← forcedIters + 1;
return S*;
```

4.3.3 Local-search operator

As mentioned earlier, tabu-search has been used to perform local-search in our ITS algorithm. In each iteration of ITS, a short run of TS is performed. In addition, the tabu-list is cleared at the beginning of each run of TS.

As mentioned in Section 4.3.1, a binary-string solution representation is used in ITS. In the TS operator, a neighbor of a given solution is obtained by adding a set to (or removing a set from) the solution.

Let X be a given configuration, S_X be the sets used in X, E_X be the elements covered by X and G_X be the groups used in X. The score of X^{+j}, which is the neighbor of X that is obtained by adding the set j to the configuration, and the score of X^{-j}, which is the neighbor of X that is obtained by removing the set j from the configuration, can be

calculated as follows :

$$\text{score}(X^{+j}) = \text{score}(X) + \sum \text{gain}(E_{X^{+j}} \backslash E_X) - c_j - \text{cost}(G_{X^{+j}} \backslash G_X) \tag{4.5}$$

$$\text{score}(X^{-j}) = \text{score}(X) - \sum \text{gain}(E_X \backslash E_{X^{-j}}) + c_j + \text{cost}(G_X \backslash G_{X^{-j}}) \tag{4.6}$$

In each iteration, the best possible non-tabu move is performed and the reverse move is added to the tabu-list ; the best move is the move that replaces the current solution with its best neighbor, i.e, the neighbor with the greatest score. The algorithm stops if the solution is not improved after $stag_{(TS)}$ iterations, where $stag_{(TS)}$ is a parameter. The tabu-list is used to avoid adding a set that was recently removed or removing a set that was recently added, and the aspiration criterion allows adding or removing such a set if the resulting solution is better than all of the solutions that have been visited so far. The pseudo-code of TS is given in Algorithm 4. The procedure *find-next-move* returns either the best non-tabu move or a tabu move that passes the aspiration criterion.

Algorithm 4 TS(S)

```
clear-tabu-lists() ;
best ← S ;
stagnationCounter ← 0 ;
loop
   m ← find-next-move(S) ;
   S ← perform-move(S, m) ;
   mark-tabu(m) ;
   if (score(S) > score(best)) then
      best ← S ;
      stagnationCounter ← 0 ;
   else if (stagnationCounter > stag(TS)) then
      return best ;
   end if
   stagnationCounter ← stagnationCounter + 1 ;
end loop
return best ;
```

A separate tabu-list is used for each type of move (namely, *add* and *remove*) to allow the use of a different tabu-list size for each type of move. Our experiments have shown that using a tabu-list for *remove* moves slows down the convergence of TS toward the optimal solution. This behavior can be explained by the fact that *remove* moves usually eliminate the redundant sets from the solution and open new possibilities for *add* moves. For this reason, *remove* moves should not be postponed for very long during the search. Instead of discarding

the tabu-list for *remove* moves, a short list was used.

4.3.4 Resulting ITS algorithm

The pseudo-code of the resulting ITS algorithm is presented in Algorithm 5.

Algorithm 5 ITS()

```
S ← empty-solution();
S* ← TS(S);
stagnationCounter ← 0;
currentGroup ← first-group();
forcedIters ← 0;
loop
   S' ← perturb1(S*);
   S*' ← TS(S');
   if (score(S*') > score(S*)) then {acceptance criterion}
      S* ← S*'
      stagnationCounter ← 0;
      S* ← release-group(S*, currentGroup); {see Section 4.3.2}
      currentGroup ← first-group();
      forcedIters ← 0;
   end if
   if (stagnationCounter > stag(ITS)) then
      S* ← perturb2(S*, forcedIters, currentGroup, stagnationCounter);
   end if
   stagnationCounter ← stagnationCounter + 1;
end loop
return S*
```

Figure 4.2 illustrates how the use of the two levels of perturbation helps ITS to reach the global optimum. The first level of perturbation (perturb_1) and the local-search operator allow the algorithm to jump from one local optimum to another in the same region until the best local optimum of the region is reached. Because the strength of the first perturbation operator is relatively small, it is difficult to reach new regions in the search space. By forcing a group into (or out of) the solution, the second perturbation operator (perturb_2) performs a long jump in the search space and reaches a new region (as shown in Figure 4.2). Then, the first perturbation operator and the local-search operator are reused to find the best local optimum of the new region. The global optimum of the problem is the best local optimum of all regions.

Figure 4.3 illustrates how the use of tabu-search as an improvement operator in ILS increases the chances of reaching the global optimum. In this case, if a simple descent heuristic

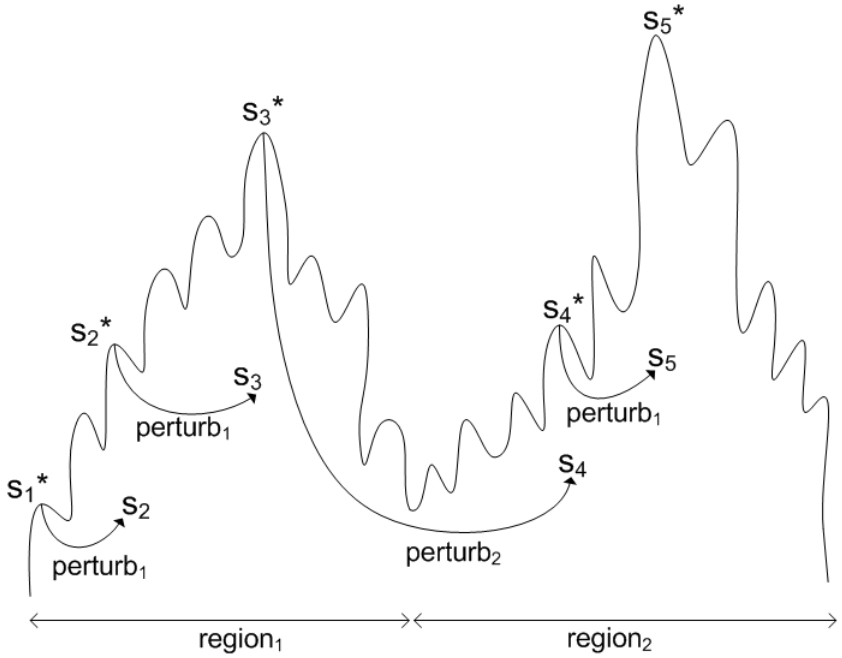

Figure 4.2 The use of two levels of perturbation increases the chances of reaching the global optimum

is used instead of tabu-search, the basin of attraction of the global optimum that is located between the two dashed lines is very small. For this reason, the probability that the perturbation operator attains the basin of attraction of the global optimum is very low and as a result, the global optimum will be difficult to reach. In contrast, tabu-search can escape from local optima located on small hills, which increases the size of the basin of attraction of the global optimum. In addition, the use of tabu-search has the effect of merging small and adjacent hills into bigger hills, which considerably reduces the total number of hills (i.e., the total number of local optima to explore). As a result, the chances of reaching the global optimum are increased as shown in Figure 4.3. On the other hand, it is important to note that the runs of tabu-search must be short enough to avoid a significant reduction of the overall number of iterations of the ILS algorithm.

4.4 The adaptation of a memetic algorithm

To evaluate the performance of the proposed algorithm, we compare it to a memetic algorithm that was proposed by [16] to solve the SCP. A memetic algorithm is a genetic algorithm (GA) that has been hybridized with an exact or heuristic algorithm [77]. Generally, a local-search-improvement heuristic is incorporated into the GA to improve the quality of each individual before it is evaluated.

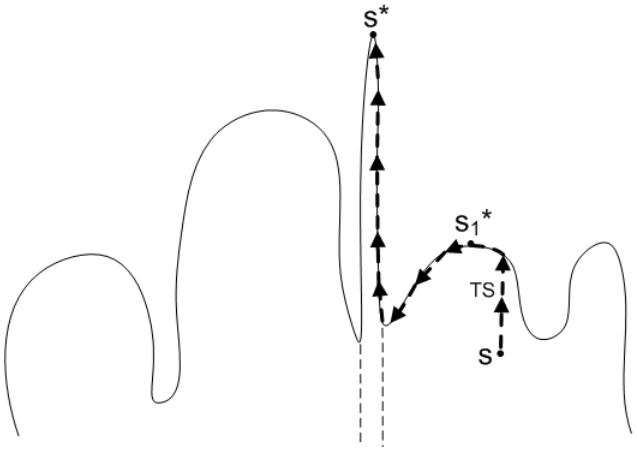

Figure 4.3 The use of tabu-search in ILS increases the probability of reaching the global optimum

We transposed as accurately as possible the memetic algorithm developed by Beasley and Chu (BCMA) from the SCP domain to the PMSCP domain. We used the same solution representation (binary string), the same way of managing the population (selection and replacement), the same crossover operator (fusion crossover), but different initialization, mutation and evaluation operators. In the following material, BCMA is briefly presented and the adapted operators are described. The reader is referred to [16] for a complete description of the original BCMA. The resulting memetic algorithm for the PMSCP is called MA-PMSCP.

In each generation of BCMA, only one child is created (i.e., BCMA is an incremental GA). Two parents are selected using the tournament selection method and combined using the fusion crossover operator. Then, the newly generated child is mutated and added to the population by replacing a randomly chosen individual, although a below-average individual is more likely to be replaced. If the newly generated child is identical to an existing member of the population, the child is discarded and a new generation is performed.

Unlike other known crossover operators (e.g., one-point, two-point and uniform), the fusion crossover operator produces only one child. When two parents are combined with the fusion operator, the resulting child has more chances to inherit genes from its fittest parent.

The initialization operator of BCMA creates a feasible solution to the SCP (i.e., all of the elements are covered) by randomly selecting a set to cover each element. For a given element, a set is randomly selected from the five cheapest sets (which are called elite sets) that cover this element. This initialization operator is not compatible with the PMSCP because the cost of a set is also influenced by the fixed cost of the group that contains the set, and as a result, the concept of elite sets is not applicable. We replaced the initialization operator of BCMA with a simple operator that randomly assigns a value of 0 or 1 to each bit.

The mutation operator of BCMA randomly flips each bit at a given mutation rate. Only elite sets are considered for mutation. Because the concept of elite sets is not applicable to the PMSCP, the adapted mutation operator considers all of the bits for mutation. As in BCMA, a variable mutation rate is used as follows : the mutation rate is low at the beginning of the search to allow high intensification using the crossover operator and is increased as the GA converges to escape from local optima.

In BCMA, a repair operator is used to ensure the feasibility of the solutions generated by the GA. In addition, a redundancy-removal operator is used to enhance the quality of the repaired solutions. The concept of feasibility is not applicable to PMSCP's because it is not necessary to cover all of the elements. We replace the repair and redundancy removal operators of BCGA with a best improvement (or steepest descent) operator that enhances the quality of the solutions produced by the genetic algorithm. The best improvement operator (BI) is executed on the newly created individuals immediately before the evaluation step of MA-PMSCP ; moreover, this operator uses the same neighborhood that is used in the tabu-search operator 4.3.3. As in TS, the best possible move is performed in each iteration. BI starts from a given configuration and performs a sequence of moves on it until the solution is locally optimal. Redundant sets are automatically removed because a configuration that contains one or more redundant sets will always have a better neighbor, which can be obtained by removing a redundant set : if X is a configuration containing a redundant set j, $X' = X - \{j\}$ is a better neighbor than X because $\text{score}(X') = \text{score}(X) + c_j$ (see equation 4.6). As a result, a solution that is improved with BI is devoid of redundant sets.

4.5 Computational results

In this section, we perform experimental analysis on the proposed iterated tabu-search algorithm and compare it to a memetic algorithm (MA-PMSCP). Experiments are performed on thirty test problems that stem from the mining industry ; these test problems contain twenty small-scale and medium-scale problems, and ten large-scale problems. The algorithms were implemented in C++ and compiled with the GNU-C++-Compiler(G++). The experiments were performed on an Intel(R)Xeon(R) X7550@2.00GHz processor with up to 1 TB of RAM.

The test problems have been generated using real geometrical data (block models and drill holes) from the mining industry. The cost associated with each set is proportional to the length of the corresponding drill hole. The gain attributed to each block is proportional to the highest cost that one would pay to cover that block. The same fixed cost is used for all groups because the cost of positioning the drill is invariable for our test problems.

The objective score of a solution has no physical interpretation and is used for comparison purposes. The characteristics of the test problems are presented in Tables 4.1 and 4.2 where the best-known solutions that are presented are the best results that were obtained in all our experiments (including CPLEX runs, which are discussed below). The density is the percentage of ones in the matrix $A^{m\times n}$ defined in Section 4.2.2.

We use the mathematical programming solver CPLEX (version 12.2) to determine the quality of the solutions found by ITS and MA-PMSCP. By running CPLEX for a sufficiently long computation time (up to nine days per instance), we obtain optimal or near-optimal solutions and an optimality gap. The optimality gap is an upper bound of the percentage deviation from the optimal solution. Some of the CPLEX runs were stopped due to RAM limitation ; which was up to 173 GB but varied for each run depending on the traffic in the test machine. The results obtained by CPLEX are presented in Table 4.3. Whereas a solution with an optimality gap of zero is optimal, a solution with an optimality gap that is close to zero is near-optimal and may in fact be optimal. The total time spend by CPLEX is presented in days (d), hours (h) or seconds (s). The tree size is the size of the branching tree used in the branch and bound algorithm of CPLEX ; this tree is stored in memory. Note that the large problems have not been solved with CPLEX. In fact, because of the size of these problems, CPLEX was not able to start branching after up to 24 hours.

Five trials of ITS and MA-PMSCP are performed for each test problem. The parameters used in ITS are as follows : $stag_{(ITS)} = 200$, $stag_{(TS)} = 1000$, $F = 50$ for small and medium-size problems, and $F = 1$ for large problems (F is decreased for large problems because the overall number of iterations is lower and as a result, the second perturbation operator is invoked fewer times). The length of the tabu-list that is used for *add* moves is equal to the number of sets in the problem that is being solved and the length of the tabu-list that is used for *remove* moves is equal to one. These parameters where chosen based on preliminary computational experiments. For MA-PMSCP, we use a population size of fifty individuals and a variable mutation rate that varies between 0.1 and 0.3. As in BCGA, the crossover operator is invoked in each generation (i.e., the crossover probability is 1).

The detailed results of ITS are presented in Tables 4.4 and 4.5. The minimum, maximum and average solutions are presented in columns $S_{(min)}$, $S_{(max)}$ and $S_{(avg)}$, respectively. The cases where ITS was able to find the optimal solution or a solution that is better than the near-optimal solution that was found by CPLEX are highlighted in bold. The other columns contain the average of the following values that correspond to a given run : T is the time in seconds when the best solution was first found, $Iters$ is the total number of iterations that were performed, Imp is the number of times the solution has been improved, $P2$ is the number of times that stagnancy has been detected in ITS (and resulted in the use of the

second perturbation operator), and $P2\text{-}Imp$ is the number of times in which the use of the second perturbation operator has allowed ITS to escape from local optima and improve the current best solution. Note that if the optimal solution is found early in a run, $P2$ will be high because the solution will be stagnant and $P2\text{-}Imp$ will be low because the solution is already optimal and cannot be improved.

In Tables 4.6 and 4.7, ITS and MA-PMSCP are compared. In addition, we experiment with two variants of ITS. The first variant (called ITS without perturb_2) is ITS without the second perturbation operator and the second variant (called ILS + perturb_2) is ITS where the TS operator has been replaced with a steepest descent operator (the steepest descent operator is similar to TS but stops when a local optimum is reached). The column $\sigma_{(avg)}$ contains the percentage deviation of the average solution from the best-known solution, the column $\sigma_{(best)}$ contains the percentage deviation of the best solution from the best-known solution and the column $T_{(avg)}$ contains the average solution-time (in seconds unless otherwise specified) for each algorithm. The total execution time for each instance is the same for all algorithms and is presented in the last column of the tables.

From the presented tables, we observe the following :

— From Tables 4.4 and 4.5, we observe that ITS performs consistently better than MA-PMSCP on all test problems (between 0.47% and 7.26% better on small-scale and medium-scale problems, and between 2.41% and 9.4% better on large-scale problems).

— From Tables 4.3 and 4.4, we observe that ITS is able to find equal or better solutions than the long CPLEX runs for 17 of the 20 small and medium-scale problems. Given the small optimality gap of CPLEX solutions and the long duration of CPLEX runs, we deduce that most of the solutions found by ITS (the bests of the five trials) are optimal or very close to optimality.

— Even though the differences between the solutions found by ITS and the solutions found by the two variants of ITS are small (Tables 4.6 and 4.7), ITS performs consistently better when both the second perturbation operator and TS are used (except for A5 where ILS + perturb_2 performs better). In addition, given the long CPLEX runs results, we observe that the use of both operators allows ITS to reach the optimal solution more frequently.

— Finally, we notice that the percentage deviations of ITS from MA-PMSCP are larger for large-scale problems than the percentage deviations that were found for small and medium-scale problems. Thus ITS is both better and more scalable than MA-PMSCP.

Table 4.1 Small and medium problems characteristics

Instance	Number of groups	Number of elements	Number of sets	Density (%)	Best-known	Element gain			Set cost			Group cost
						min	max	avg	min	max	avg	
A1	10	1000	3493	2.1	150386	100	300	199.1	400	1300	824.4	1000
A2	10	1000	4300	5.2	179973	100	300	199.1	400	1300	820.4	1000
A3	10	1000	6657	2.2	155266	100	300	199.1	400	1300	825.0	1000
A4	10	1000	13076	2.2	156558	100	300	199.1	400	1300	824.9	1000
A5	10	1000	82635	2.5	160688	100	300	199.1	300	1500	794.2	1000
B1	20	1000	6924	2.1	152335	100	300	199.1	400	1300	824.3	1000
B2	20	1000	10617	2.2	155554	100	300	199.1	400	1300	824.2	1000
B3	20	1000	26043	2.2	157551	100	300	199.1	400	1300	824.9	1000
B4	20	1000	40351	2.3	158403	100	300	199.1	400	1300	824.0	1000
B5	20	1000	70899	2.4	159426	100	300	199.1	400	1300	824.3	1000
C1	31	5192	6633	3.3	246121	1	100	50.8	100	1200	542.5	1000
C2	31	5192	12088	3.4	247455	1	100	50.8	100	1200	539.0	1000
C3	31	5192	24007	3.4	249070	1	100	50.8	100	1200	540.9	1000
C4	31	5192	36689	3.4	249124	1	100	50.8	100	1200	540.2	1000
C5	31	5192	63635	3.4	249935	1	100	50.8	100	1200	539.4	1000
E1	62	5192	15252	2.9	247199	1	100	50.8	100	1200	502.7	1000
E2	62	5192	28303	3.0	248389	1	100	50.8	100	1200	499.8	1000
E3	62	5192	55975	3.0	248923	1	100	50.8	100	1200	500.4	1000
E4	62	5192	85978	2.5	249107	1	100	50.8	100	1200	499.9	1000
E5	62	5192	177274	3.1	250158	1	100	50.8	100	1200	493.0	1000

Table 4.2 Large problems characteristics

Instance	Number of groups	Number of elements	Number of sets	Density (%)	Best-known	Element gain			Set cost			Group cost
						min	max	avg	min	max	avg	
E1	100	15000	8521	1.18	619800	1	100	50.64	1500	1500	1500	1000
E2	100	15000	16264	0.55	449288	1	100	50.64	1500	1500	1500	1000
E3	100	15000	30673	0.73	548110	1	100	50.64	1500	1500	1500	1000
E4	100	15000	57508	0.93	637801	1	100	50.64	500	1500	854.18	500
E5	100	15000	67368	0.5	472158	1	100	50.64	1000	1500	1163.36	1000
F1	100	15000	86308	0.57	500797	1	100	50.64	1000	1500	1139.15	500
F2	100	15000	105959	0.4	494333	1	100	50.64	500	1500	890.41	500
F3	100	15000	148104	0.38	485309	1	100	50.64	500	1400	856.76	1000
F4	100	15000	239331	0.5	495066	1	100	50.64	1000	1500	1165.41	1000
F5	100	15000	408690	1.17	484835	1	100	50.64	500	1400	857.34	1000

Table 4.3 Optimal or near-optimal solutions obtained with CPLEX

Instance	Solution	Optimality gap (%)	Total time	Tree size
A1	150386	0	6.3 (h)	250 MB
A2	179973	0	450 (s)	50 MB
A3	155242	0.51	4.6 (d)	96.3 GB
A4	156558	1.28	2.9 (d)	173.2 GB
A5	158550	5.23	8.5 (d)	124.7 GB
B1	152222	1.73	3.2 (d)	135.9 GB
B2	155310	2.88	2.75 (d)	95.0 GB
B3	156509	5.01	3.2 (d)	89.2 GB
B4	156851	6.26	1.3 (d)	113.0 GB
B5	158903	6.11	7.2 (d)	48.4 GB
C1	246121	0	1933 (s)	27.8 MB
C2	247455	0	6.3 (h)	229.9 MB
C3	249070	0	3.7 (h)	188.5 MB
C4	249124	0	23.6 (h)	2.0 GB
C5	249935	0	16.7 (h)	306.4 MB
D1	247112	0.43	7.2 (d)	82.1 GB
D2	248200	0.82	7.8 (d)	32.1 GB
D3	248136	1.6	8.2 (d)	9.4 GB
D4	247809	2.01	8.0 (d)	4.3 GB
D5	249929	1.59	9.0 (d)	36.9 GB

Table 4.4 Detailed results for ITS on small and medium problems

Instance	$S_{(min)}$	$S_{(max)}$	$S_{(avg)}$	$T_{(avg)}$(s)	$Iters_{(avg)}$	$Imp_{(avg)}$	$P2_{(avg)}$	$P2\text{-}Imp_{(avg)}$
A1	150158	**150386**	150245.6	159	20425.4	14.6	30	2.2
A2	179973	**179973**	179973	122	9350.8	9.4	14	1.6
A3	154150	**155266**	154909.2	618.8	9729.8	28	14.2	1.8
A4	156243	156423	156338.2	480.2	4846.6	23.4	6.8	1
A5	159673	**160276**	159963	1141.4	1751	31.6	2.2	0.6
B1	152110	**152335**	152290	79.6	9830.2	15.6	9.2	1.6
B2	155409	**155554**	155524.2	278.2	6617.2	18	6	1
B3	157249	**157551**	157455	904.6	2417.4	23.8	2.8	1.8
B4	157872	**158403**	158215	983.6	1710.6	21.6	2.4	1.4
B5	158298	**159426**	158827	810.8	1033.6	20.2	1	0.4
C1	246121	**246121**	246121	10.4	5417.2	19.2	3	0
C2	247447	**247455**	247450.2	355.2	2774.4	16.2	3.4	2.2
C3	249070	**249070**	249070	93.6	1671	24.8	2	1
C4	249060	249094	249072.4	543.8	1442.2	22.2	1.8	1
C5	249501	249881	249756.8	1763	1968.6	20.6	2.6	1.8
D1	247107	**247199**	247170.6	585.2	2764.8	27.4	3.4	2.6
D2	247723	**248389**	247968.6	984.8	1318	31.4	1	0.2
D3	248474	**248923**	248663.8	2468.2	1223.6	25.6	1.6	1
D4	248107	**249107**	248672.2	5376	1404	34.8	1.8	1
D5	249605	**250158**	249956.8	3439.6	1418.4	31.6	1.2	0.6

Table 4.5 Detailed results for ITS on large problems

Instance	$S_{(min)}$	$S_{(max)}$	$S_{(avg)}$	$T_{(avg)}$(s)	$Iters_{(avg)}$	$Imp_{(avg)}$	$P2_{(avg)}$	$P2\text{-}Imp_{(avg)}$
E1	617248	619800	618852	12806.8	56810.6	34.6	186.4	1.2
E2	445572	449288	446976.2	15965.6	61644.6	63.2	196.8	2.8
E3	543841	548110	545262.2	14750.4	23383.4	44.2	71.2	1.2
E4	635294	637010	636065.4	28585.2	9321.2	46	27.4	8.4
E5	468404	472158	470955.4	28794	12116.6	47.4	32.8	1.6
F1	496208	500797	498983.6	31044	8717.8	44.8	25	6.8
F2	487641	494333	490600.4	32674.2	8240	42.8	24	7.4
F3	481691	485309	483177.8	34616	5317.8	39.6	14	6.8
F4	480028	484835	481769.8	26312.8	2222.2	26	5.4	3.4
F5	492514	495066	494008.2	34696.4	1424.2	20	3.2	3.2

4.6 Conclusions and future work

In this work, we proposed a new variant of the partial set covering problem, a mining-industry application to it and a heuristic algorithm to solve it. The analysis of our computational results shows that the proposed algorithm is very effective and scalable for solving the PMSCP. We showed that most of the solutions found by ITS are either optimal or very close to optimality.

In addition, we theoretically discussed how the use of tabu-search and a second perturbation operator in iterated-local-search can help escape from the local optimum located on small hills and the local optima that have large basins of attraction. Furthermore, we experimentally showed that the use of both the second perturbation operator and tabu-search allows the algorithm to reach the optimal solution more frequently.

We end this paper by motivating the use of multiple perturbation operators in ILS. Even though we used two perturbation operators to target the sets and the groups of the PMSCP, other applications can benefit from the use of more perturbation operators. In addition, each perturbation operator can have a frequency that is dependent on the strength of the perturbation ; where the frequency is used to determine how frequently an operator will be invoked. For instance, in our case, the first perturbation operator is invoked in each iteration (i.e. has a frequency of one) while the second perturbation operator (that has a higher strength) is invoked in each $stag_{(ITS)}$ iterations (i.e. has a frequency of $stag_{(ITS)}$). Similarly, the use of multiple local-search operators in ILS can be an interesting subject for future research.

Table 4.6 Comparison of ITS, ITS without perturb$_2$, ILS + perturb$_2$ and MA-PMSCP on small and medium problems

	ITS			ITS without perturb$_2$			ILS + perturb$_2$			MA-PMSCP			
Instance	$\sigma_{(avg)}$	$\sigma_{(best)}$	$T_{(avg)}$	$\sigma_{(avg)}$	$\sigma_{(best)}$	$T_{(avg)}$	$\sigma_{(avg)}$	$\sigma_{(best)}$	$T_{(avg)}$	$\sigma_{(avg)}$	$\sigma_{(best)}$	$T_{(avg)}$	$T_{(tot)}$
A1	0.09	0	159	0.26	0.15	938	0.09	0	399.2	1.13	0.86	647.0	1800
A2	0	0	122	0	0	670	0.01	0	419	0.47	0.38	1434.0	1800
A3	0.23	0	618.8	0.39	0	204.4	0.24	0	234.2	3.28	3.05	619.8	1800
A4	0.14	0.09	480.2	0.38	0.23	493	0.3	0.13	560	4.21	3.72	837.0	1800
A5	0.45	0.26	1141.4	0.68	0.4	2972.4	0.34	0	1196.4	5.92	5.5	1129.0	3600
B1	0.03	0	79.6	0	0	476.4	0.07	0.07	979.4	2.91	2.42	483.0	1800
B2	0.02	0	278.2	0.14	0.06	787.4	0.11	0	414.4	4.53	4.18	600.0	1800
B3	0.06	0	904.6	0.57	0.19	982.6	0.28	0.14	711.8	5.7	4.34	529.8	1800
B4	0.12	0	983.6	0.65	0.3	508.4	0.21	0.08	28.4	6.87	6.3	682.8	1800
B5	0.38	0	810.8	0.83	0.53	1353.8	0.47	0.39	653.4	7.26	6.91	538.6	1800
C1	0	0	10.4	0	0	548.2	0	0	307.4	2.82	2.52	1301.4	1800
C2	0	0	355.2	0.02	0	853.6	0.03	0.01	1220	3.29	2.8	1195.2	1800
C3	0	0	93.6	0.09	0	616.4	0.08	0.03	609.8	3.51	3.33	834.0	1800
C4	0.02	0.01	543.8	0.11	0.01	859.6	0.12	0.06	1644.6	3.95	3.48	1182.6	1800
C5	0.07	0.02	1763	0.37	0.35	52	0.1	0.03	15	4.38	4.29	1707.2	3600
D1	0.01	0	585.2	0.06	0.01	6271.2	0.07	0.03	2788.6	2.1	1.88	1043.8	1800
D2	0.17	0	984.8	0.22	0.03	583.8	0.22	0.11	723.8	2.87	2.68	622.0	1800
D3	0.1	0	2468.2	0.25	0.13	1280.4	0.15	0.05	1241.2	3.68	3.28	1611.8	3600
D4	0.17	0	5376	0.43	0.25	2159.4	0.43	0.23	1646	3.29	3.15	889.8	7200
D5	0.08	0	3439.6	0.43	0.19	4372.8	0.21	0.03	3723.4	3.64	3.56	4809.8	7200

Table 4.7 Comparison of ITS, ITS without perturb$_2$, ILS + perturb$_2$ and MA-PMSCP on large problems

	ITS			ITS without perturb$_2$			ILS + perturb$_2$			MA-PMSCP			
Instance	$\sigma_{(avg)}$	$\sigma_{(best)}$	$T_{(avg)}$	$\sigma_{(avg)}$	$\sigma_{(best)}$	$T_{(avg)}$	$\sigma_{(avg)}$	$\sigma_{(best)}$	$T_{(avg)}$	$\sigma_{(avg)}$	$\sigma_{(best)}$	$T_{(avg)}$	$T_{(tot)}$
E1	0.15	0	3.56 h	0.27	0.2	1.94 h	0.23	0.01	2.42 h	2.86	2.41	2.66	10 h
E2	0.51	0	4.43 h	0.54	0.3	3.53 h	0.6	0.07	4.97 h	6.13	5.99	4.46	10 h
E3	0.52	0	4.1 h	0.47	0.19	6.01 h	0.75	0.52	8.45 h	5.39	4.91	5.36	10 h
E4	0.27	0.12	7.94 h	0.22	0	7.58 h	0.53	0.28	8.54 h	3.66	3.61	7.26	10 h
E5	0.25	0	8 h	0.43	0.34	7.57 h	1.09	0.83	9.47 h	8.35	8.04	8.88	10 h
F1	0.36	0	8.62 h	0.63	0.32	9.44 h	1.45	0.91	7.67 h	6.61	6.4	6.12	10 h
F2	0.76	0	9.08 h	0.89	0.72	7.98 h	1.5	1.15	8.59 h	5.57	5.46	1.82	10 h
F3	0.44	0	9.62 h	0.84	0.46	8.49 h	0.91	0.44	8.50 h	9.26	8.92	5.31	10 h
F4	0.63	0	7.31 h	0.93	0.47	7.84 h	1.61	0.87	4.17 h	9.57	9.4	8.69	10 h
F5	0.21	0	9.64 h	0.26	0.08	7.44 h	1.84	1.48	7.56 h	9.02	8.84	5.53	10 h

CHAPITRE 5

ARTICLE 2 : A NEW FORMULATION OF THE SET COVERING PROBLEM FOR METAHEURISTIC APPROACHES

Auteurs : Nehme Bilal, Philippe Galinier et Francois Guibault
Revue : *ISRN Operations Research*
Référence : [19]

Abstract

Two difficulties arise when solving the set covering problem (SCP) with metaheuristic approaches : solution infeasibility and set redundancy. In this paper, we first present a review and analysis of the heuristic approaches that have been used in the literature to address these difficulties. We then present a new formulation that can be used to solve the SCP as an unconstrained optimization problem and that eliminates the need to address the infeasibility and set redundancy issues. We show that all local optimums with respect to the new formulation and a 1-flip neighbourhood structure are feasible and free of redundant sets. In addition, we adapt an existing greedy heuristic for the SCP to the new formulation and compare the adapted heuristic to the original heuristic using 88 known test problems for the SCP. Computational results show that the adapted heuristic finds better results than the original heuristic on most of the test problems in shorter computation times.

5.1 Introduction

The Set Covering Problem (SCP) is a popular optimization problem that has been applied to a wide range of industrial applications, including scheduling, manufacturing, service planning and location problems [22, 61, 6, 25]. The SCP is *NP-hard* in the strong sense [42]. The mathematical formulation of the SCP is as follows. Let $E = \{e_1, ..., e_m\}$ be a universe of elements and $S = \{s_1, ..., s_n\}$ be a collection of subsets $s_j \subseteq E$, where $\bigcup s_j = E$. Each set s_j covers at least one element of E and has an associated cost $c_j > 0$. The objective is to find a subcollection of sets $X \subseteq S$ that covers all of the elements in E at a minimal cost. The mathematical programming model of the SCP is usually formulated as follows. Let

— $A^{m \times n}$ be a zero-one matrix where $a_{ij} = 1$ if element i is covered by set j, and $a_{ij} = 0$ otherwise.

— $X = \{x_1, x_2, ..., x_n\}$ where $x_j = 1$ if set j (with cost $c_j > 0$) is part of the solution, and $x_j = 0$ otherwise.

Minimize

$$\sum_{j=1}^{n} c_j x_j \tag{5.1}$$

subject to

$$1 \leq \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, ..., m \tag{5.2}$$

$$x_j \in \{0, 1\} \tag{5.3}$$

The objective function (5.1) drives the search toward solutions at minimal cost. Constraint (5.2) (full coverage constraint) imposes the requirement that all the elements of the universe E must be covered. If constraint (5.2) is not satisfied, the solution is infeasible. If constraint (5.2) is satisfied and the objective function is minimized, the solution will cover all of the elements at the minimal cost (optimal solution). If constraint (5.2) is relaxed, the objective function will drive the search toward an empty solution because the empty solution has the lowest cost (0). These observations show that the objective function and the full coverage constraint of the SCP guide the search in two opposite directions.

When solving the model with metaheuristic algorithms, two issues arise : solution infeasibility and set redundancy. A solution to the SCP is considered to be infeasible if one or more elements of the universe E are uncovered. A set is considered to be redundant if all the elements covered by the set are also covered by other sets in the solution.

In this paper, we first review and analyze the literature to highlight the difficulties in dealing with solution infeasibility and set redundancy when solving the SCP with metaheuristic algorithms (section 5.2). We then present a new formulation that can be used to solve the SCP as an unconstrained optimization problem and that eliminates the need for addressing the infeasibility and redundancy issues (Section 5.3). The new formulation uses a maximization objective that can replace both the cost minimization objective and the full coverage constraint of the classical formulation. The new formulation can also be seen as a new penalty approach that has many advantages over existing penalty approaches (5.3.2) for the SCP. Third, we present a simple descent heuristic that is based on the new formulation and that uses a simple 1-flip neighbourhood structure. The proposed descent heuristic is an adaptation of an existing greedy heuristic for the SCP. We show that all local optimums with respect to the new formulation and the 1-flip neighbourhood structure are feasible and free of redundant sets. Finally, the proposed descent heuristic is compared to the original greedy

heuristic using 88 known set covering problems (section 5.5).

5.2 Literature Review

In general, metaheuristic algorithms can be divided into three categories :

— *Constructive metaheuristics* : In each iteration, a new local optimum is found by constructing a new solution from scratch. A level of randomness is added to the construction step in order to avoid constructing the same solution over and over.

— *Evolutionary algorithms* : In each iteration, two or more solutions are combined to create an new solution.

— *Local search* : In each iteration, the current solution is replaced by one of its immediate neighbors (the solution is usually modified slightly).

In the following sections, we review the literature of solving the SCP with metaheuristic approaches and analyze how each category of metaheuristics addresses *solution infeasibility* and *set redundancy*.

5.2.1 Constructive metaheuristics

When the SCP is solved with constructive metaheuristics, the local optimums found at the end of each constructive iteration are usually feasible. In fact, the constructive iteration ends when all of the elements are covered. For this reason, these metaheuristics do not have to deal with the infeasibility issue. However, the local optimums are not necessarily free of redundant sets, and a redundancy removal heuristic is needed. Constructive metaheuristics for the SCP includes ant colony optimization [63, 78, 79, 81, 31], Meta-RaPS [34] and GRASP [36]. All of these metaheuristics use a dedicated redundancy removal operator that removes redundant sets at the end of each iteration.

5.2.2 Evolutionary algorithms

Evolutionary algorithms for the SCP need to address both infeasibility and set redundancy issues. Most evolutionary algorithms that are used to solve the SCP are based on the genetic algorithm (GA). Most GA's use a binary string solution representation where $x_j = 1$ if the set s_j is part of the solution, and $x_j = 0$ otherwise. The infeasibility issue arises when the crossover or mutation operator of the GA produces a child (solution) that does not cover all the elements. In fact, a simple bit flip from 1 to 0 during crossover or mutation can produce an infeasible solution. If a cost minimization objective function is used, infeasible solutions will be preferred over feasible ones because infeasible solutions are usually cheaper. Two main approaches have been used in the literature to address the infeasibility issue.

The first approach uses a repair heuristic to transform infeasible solutions to feasible solutions before the evaluation step of the GA. A greedy-like repair heuristic is usually used [16, 84, 74]. In each iteration, the repair operator covers an uncovered element by selecting a new set that covers the element and adding it to the solution. In [74], all of the solutions are repaired for evaluation but only 5% of them are replaced with the corresponding repaired versions. The aim is to allow the search to explore infeasible regions of the search space, which tend to be more effective than limiting the search to only feasible regions. A simpler repair heuristic is used in [5]. During the evaluation of a solution, a set is added to the solution if it covers an uncovered element(s) and is not already part of the solution. By adding new sets, repair heuristics may introduce redundant sets into the solutions. For this reason, genetic algorithms that use a repair operator also use a redundancy removal procedure that is applied after the repair and just before evaluation.

The second approach involves penalizing the objective value of infeasible solutions to drive the search toward the feasible region. A penalty term that makes infeasible solutions less attractive than feasible ones is added to the objective function. In [3], the same penalty M is added to the objective value of all infeasible solutions. M is high enough to guarantee that all feasible solutions have lower objective values than all infeasible solutions ($M = \sum c_j$). A drawback of using such an objective function is that infeasible solutions cannot be compared to each other because the objective function does not reflect the degree of infeasibility. Objective functions that penalize infeasible solutions while reflecting the degree of infeasibility are proposed in [90] and [5]. In [90], the penalty attributed to an infeasible solution is proportional to the number of elements that are not covered in the solution. In [5], the penalty is proportional to the minimum cost it would take to cover all of the uncovered elements. In all discussed penalty approaches, the penalties are high enough to ensure that all infeasible solutions have higher objective values than all feasible ones. An immediate disadvantage of using such high penalties is that feasible solutions will always be preferred over infeasible ones. As a result, infeasible solutions will have low chances of surviving in the population, and the infeasible region of the search space will not be effectively explored.

5.2.3 Local search

The feasibility constraint makes designing an effective local search metaheuristic for the SCP a difficult task. For this reason, few local search only heuristics have been developed for the SCP [72, 79]. Instead, most of the local search algorithms have combined local search with other techniques such as *Lagrangian relaxation*, *subgradiant optimization*, *group theory* and *linear programming* [92, 7, 14, 22, 55, 56]. In [13], after noting the difficulty of defining a good neighbourhood to solve the unicost set covering problem with local search, the authors

proposed that the problem could be transformed to an equivalent satisfiability problem (SAT) that can be solved more adequately with local search.

Most local search algorithms for the SCP use a simple 1-flip neighbourhood structure defined by moves that only add (remove) one set at a time to (from) the solution. When a local optimum is reached, which is usually a feasible solution, it is difficult to decide in which direction to continue the search. Two cases arise :

(i) If the search space is restricted to the feasible region, only redundant sets are allowed to be removed. If no redundant sets exist in the solution, at least one redundant set must be added before a *remove* move is allowed to be performed. As a result, the infeasible region of the search space will not be explored and the search will tend to fall into local optimums and cycles. A more complex neighbourhood called 3-flip is used in [79] to make the search in the feasible region more effective. The 3-flip neighbourhood of a given solution consists of all of the solutions that can be obtained by adding (removing) at most 3 sets to (from) the solution. Even though the proposed heuristic is more effective than a simple 1-flip heuristic, it is not sufficient to avoid local optimums and cycles and it is significantly slower than the 1-flip heuristics.

(ii) If the search space is not restricted to the feasible region, the cost minimization objective drives the search toward the infeasible region, by removing sets from the current configuration (to minimize the cost), and it is unclear when to restore feasibility. In such situations, penalty approaches are usually used to penalize infeasible solutions.

If the penalty weights are too high, neighbors in the feasible region will be preferred over neighbors in the infeasible region, making the infeasible region unreachable. Lower or dynamic penalty weights are usually used to make the search more effective by allowing it to reach infeasible regions.

If the penalty weights are too low, the final solution found is not guaranteed to be feasible. A tabu search heuristic that uses such low penalties is proposed in [72] for the unicost set covering problem. A simple 1-flip neighbourhood structure is used. The objective is to minimize $(C + E)$ where C is the number of sets used in the solution, and E is the number of uncovered elements. If a set covers only one uncovered element, adding (removing) it to (from) the solution will not have any effect on the objective function. As a result, this set might be left out of the solution, making it infeasible. To overcome the fact that this objective function does not guarantee feasibility, the neighbourhood is restricted such that if a set is removed during one iteration, one or more sets must be added in the next iteration to restore feasibility. Even though such a low penalty approach allows the search to reach the infeasible regions, additional neighbourhood restrictions are used to restore feasibility, and the infeasible region is only scratched.

Dynamic penalty approaches, in which the penalty weights are repeatedly adjusted, are used to balance the search between the feasible and infeasible regions without using a repair operator or neighbourhood restrictions [92, 24, 7, 14, 22]. The most frequent dynamic penalty approaches that have been used in the literature are based on Lagrangian relaxation [37] and subgradiant optimization [49]. Dynamic penalty approaches can be very effective but are difficult to design and implement.

5.3 Proposed Formulation

In this work, we propose a new formulation of the SCP with a maximization objective. The aim of the proposed formulation is to express the real objective of the SCP in the objective function which is to *cover all* elements at a *minimal cost*. We view covering an element as collecting a gain at a given cost. In this perspective, we attribute a gain to each element. Because all elements must be covered, the gain attributed to each element must be higher than the cost of at least one of the sets that covers the element ; otherwise, there is no benefit of covering that element. Let $c_{min}(e_i)$ be the cost of the cheapest set among the sets that cover the element e_i. A gain $g_i = c_{min}(e_i) + \epsilon$ is attributed to each element e_i where ϵ is a small positive constant.

Let

— $A^{m \times n}$ be a zero-one matrix where $a_{ij} = 1$ if element e_i is covered by set j, and $a_{ij} = 0$ otherwise.

— $X = \{x_1, x_2, ..., x_n\}$ where $x_j = 1$ if set s_j (with cost $c_j > 0$) is part of the solution, and $x_j = 0$ otherwise.

— $Y = \{y_1, y_2, ..., y_m\}$ where $y_i = 1$ if element e_i (with gain $g_i > 0$) is covered in the solution, $y_i = 0$ otherwise.

Maximize

$$\sum_{i=1}^{m} g_i y_i - \sum_{j=1}^{n} c_j x_j \tag{5.4}$$

subject to

$$y_i \leq \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, ..., m \tag{5.5}$$

$$x_j, y_i \in \{0, 1\} \tag{5.6}$$

Constraint (5.5) is a relaxation of constraint (5.2) because it does not impose coverage of all the elements ; its only purpose is to keep track of which elements of E are part of the cover.

Constraint (5.6) is the integrity constraint in mathematical programming. Constraints (5.5) and (5.6) need not to be addressed as constraints in heuristic approaches but are presented for completeness of the mathematical programming formulation.

Claim 1 : The optimal solution of the proposed formulation is a feasible solution (covers all elements).

Proof : Suppose that the optimal solution does not cover all of the elements and has an objective value P. Let e_i be an uncovered element. By the definition of the gain g_i, we know that there is at least a set s_j that covers element e_i and has a cost $c_j = c_{min}(e_i) = g_i - \epsilon$. If the set s_j is added to the cover, the new objective value is $P' = P + (g_i - c_j) = P + (g_i - (g_i + \epsilon)) = P + \epsilon > P$. Thus, P is not optimal. By contradiction, we conclude that the optimal solution covers all of the elements.

Claim 2 : The optimal solution of the proposed formulation covers all elements at a minimal cost.

Proof : We proved in *claim 1* that the optimal solution covers all of the elements. Hence, the first term of the objective function (5.4) is a constant in the optimal solution ($\sum_{i=1}^{m} g_i y_i = K$). The objective function becomes :

$$\begin{aligned} \text{Maximize } (K - \sum_{j=1}^{n} c_j x_j) &\Rightarrow \text{Maximize } (-\sum_{j=1}^{n} c_j x_j) \\ &\Rightarrow \text{Minimize } \sum_{j=1}^{n} c_j x_j \end{aligned}$$

Thus, the optimal solution of the proposed formulation is the cheapest feasible solution, which is the objective of the SCP.

From heuristic algorithms perspective, we replaced a constrained optimization problem with an unconstrained optimization problem that has the same optimal solutions. Unconstrained optimization problems are known to be much easier to solve with heuristic algorithms than constrained optimization problems.

5.3.1 Comparison to penalty approaches

Even though the proposed formulation is a full mathematical programming formulation for the SCP, it is similar to existing penalty approaches ; but with some important differences. The objective function presented in equation (5.4) can be rewritten as :

$$\text{Minimize } \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} g_i \overline{y_i} \tag{5.7}$$

where $\overline{y_i} = 1$ if element e_i is uncovered, and $\overline{y_i} = 0$ otherwise. The value of the gain g_i can be seen as the penalty associated with not covering the element e_i.

The proposed approach is different from high penalty approaches because some infeasible solutions might have a better objective value than some feasible ones. For instance, let $U = \{s_1, s_2, s_3\}$, $s_1 = \{e_1, e_2, e_3\}$, $s_2 = \{e_1, e_2\}$ and $s_3 = \{e_3\}$. The cost of the sets are $c_1 = 10$, $c_2 = 2$ and $c_3 = 1$. The cheapest set that covers the element e_1 is s_2 with a cost $c_2 = 2$. Thus, by definition of the gain, g_1 is equal to $c_2 + \epsilon = 2 + \epsilon$. Similarly, we find $g_2 = 2 + \epsilon$ and $g_3 = 1 + \epsilon$. Let $X_1 = \{s_1\}$ be a feasible solution and $X_2 = \{s_2\}$ be an infeasible one. Using the objective function (5.7), the objective value of X_1 is 10 and the objective value of X_2 is $(c_2 + e_3) = (3 + \epsilon)$. Thus, the infeasible solution X_2 has a lower (better) objective value than the feasible solution X_1, which does not occur with high penalty approaches.

The proposed approach is different from low penalty approaches because the penalties are high enough to drive the search toward the feasible region. We showed that the optimal solutions with respect to the new formulation are guaranteed to be feasible. The proof of feasibility of the optimal solution also shows that any infeasible solution can be transformed to a feasible one with a better objective value. For instance, in the previous example, the infeasible solution X_2 can be transformed to a feasible solution $X_3 = \{s_2, s_3\}$ (by adding the set s_3 to X_2) with an objective value of 3, which is lower (better) than the objective value of X_2 $(3 + \epsilon)$.

The proposed penalty approach is different from dynamic penalty approaches because the penalty weights are static and no adjustment is needed.

When high penalty approaches are used, the search process of a heuristic algorithm is disturbed by the high penalties and driven immediately to the feasible region. On the other hand, low penalties do not disturb the search but cannot ensure feasibility. The aim of our approach is to choose the lowest possible penalties that avoid disturbing the search process while ensuring feasibility. Ensuring feasibility means that any infeasible solution can be transformed to a feasible one with a better objective value.

5.3.2 Benefits of the new formulation with respect to metaheuristics

The new formulation eliminates all issues related to solution infeasibility and set redundancy that were discussed in the literature review (section 5.2). Because the objective function naturally penalizes redundant sets, the use of a redundancy removal operator is not needed. The objective function also penalizes infeasible solutions. As a result, the use of a repair or penalty approaches in evolutionary algorithms and the use of neighbourhood restrictions in local search algorithms are not needed. Finally, because no constraints are involved and the only driver of the search is the objective function proposed with the new formulation,

designing a good neighbourhood and local search algorithm is quite simple. Such a simple neighbourhood is presented in section 5.4.

5.4 Proposed Descent Heuristic (DH)

In this section, we present a simple descent heuristic that is based on the new formulation and that uses a 1-flip neighbourhood structure. We also show that all local optimums with respect to the new formulation and the 1-flip neighbourhood are feasible and free of redundant sets.

The proposed descent heuristic (DH) is an adaptation of the classical greedy heuristic that has been used in the literature for the SCP [28]. In this greedy heuristic, the set s_j with the minimum ratio $\eta_j = c_j/card_j(X)$ is added to the solution in each iteration. The term $card_j(X)$ is the number of elements that are covered by s_j and are not covered by the current configuration X. Once all of the elements are covered, redundant sets are removed in decreasing order of cost. In DH, the term $card_j(X)$ of the classical greedy heuristic is replaced with δ_j, where δ_j is the variation in the objective function associated with adding (removing) the set s_j.

DH starts from a given configuration and performs a sequence of moves on it until the solution is locally optimal. It uses a simple 1-flip neighbourhood structure with two types of moves : *add* and *remove* moves. $add(j)$ adds the set s_j to the configuration (flips x_j from 0 to 1), while $remove(j)$ removes the set s_j from the configuration. In each iteration, the set s_j with the maximum ratio $R_j = \delta_j/c_j$ is added (removed) to (from) the solution. The algorithm stops when the current configuration is better than all of it's neighbors ($R_j \leq 0 \ \forall j$). The outline of DH is presented in Algorithm 6.

Algorithm 6 DH()

```
sol ← empty solution ;
loop
  find the set s_j with the maximum ratio R_j ;
  if (R_j > 0) then
    flip bit x_j ;
  else
    stop ;
  end if
end loop
```

5.4.1 Redundancy removal

In contrast to the classical greedy heuristic, DH automatically removes the redundant sets from the solution. Let X be a configuration where the set s_j is redundant. The ratio R_j associated with removing s_j from X is equal to $(c_j - 0)/c_j = 1$. Because $R_j > 0$, the move $remove(j)$ will be performed, and the redundant sets will be removed. As a result, any solution that is improved with DH is necessarily free of redundant sets. The redundant sets are removed at any time during the progress of DH and not only at the end.

5.4.2 Feasibility

Consider X to be a configuration where e_i is not covered. Let s^i_{min} be the cheapest set that covers e_i and c^i_{min} be its associated cost. The gain g_i associated with e_i is equal to $c^i_{min} + \epsilon$. If e_i is the only uncovered element covered by s^i_{min} (worst case scenario), the ratio R^i_{min} associated with adding the set s^i_{min} to X is equal to $(c^i_{min} + \epsilon - c^i_{min})/c^i_{min} = \epsilon/c^i_{min}$. Because $R^i_{min} > 0$ $(\forall \epsilon > 0)$, the move $add(s^i_{min})$ will be performed and the solution will be feasible. As a result, any solution that is improved with DH is feasible.

5.4.3 Discussion

We showed that all of the solutions that are found with DH are feasible and free of redundant sets. With respect to the new formulation and the 1-flip neighbourhood structure, these solutions are local optimums. This is also true for all solutions obtained with any descent heuristic that is based on the new formulation and that uses the same neighbourhood structure. As a result, all local optimums with respect to the new formulation and the 1-flip neighbourhood structure are feasible and free of redundant sets

5.5 Experimental Analysis

In this section, we present computational experiments with the proposed descent heuristic that is based on the new formulation. Although we showed in the previous sections that the new formulation provides many advantages over the classical formulation, the final performance of any metaheuristic algorithm depends on the implementation, the tuning of the parameters and the sophistication of the approach. We do not assume that any metaheuristic approach that is based on the new formulation will outperform all metaheuristic approaches that are based on the classical formulation. In addition, experimenting with all classes of metaheuristics will not prove (or disprove) the superiority of the proposed formulation. Instead, we compare our descent heuristic to the original greedy heuristic that is based on the

classical formulation. The aim is to compare the two formulations using similar algorithms. Since greedy heuristics are used for intensification in most metaheuristic approaches for the SCP, evaluating the effectiveness of a new descent heuristic that can replace these greedy heuristics provides a good indication of how suitable is the new formulation to metaheuristic approaches.

We compare DH to the classical greedy heuristic (GH) [28] on three classes of known set covering problems.

— **OR-Library benchmarks :** This class includes 65 small and medium size randomly generated problems that were frequently used in the literature. Most metaheuristic approaches for the SCP have been tested on these problems. They are available in OR-Library [15] and are described in table 5.1.

— **Airline and bus scheduling problems :** This class includes fourteen real world airline scheduling problems (AA instances) and two bus driver scheduling problems (BUS instances). These problems were obtained from [93] and are described in table 5.2.

— **Railway scheduling problems :** This class includes seven large-scale railway crew scheduling problems from Italian railways and are available in OR-Library [15]. These problems are described in table 5.3.

Most metaheuristic approaches for the SCP have been exclusively tested on OR-Library benchmarks. Because these benchmarks are relatively small, we experimented with larger problems that have been less frequently used in the literature.

In all presented tables, the name of each instance is given in the first column, the size of each instance is given in the second column (number of elements $\times$ number of sets) and the density of each instance is given in the third column. The density is the percentage of ones in the $A^{m \times n}$ matrix described in Section 5.1). The optimal or best-known solution of each instance is given in the fourth column. The solutions obtained with each heuristic are presented in columns 5 and 6. The last two columns contain the number of iterations performed by each heuristic for each instance. The percentage deviations from the best-known solutions are presented in figures 5.1, 5.2, 5.3 and 5.4.

In both DH and GH, each iteration involves finding the best set to add (remove) to (from) the solution and updating the underlying data-structure after a move is performed. Thus, the algorithmic complexity of each iteration is similar in both heuristics. In practice, the computation times are highly dependent on the implementation and the characteristics of the problem solved (size and density). For instance, finding the best move to be performed in each iteration can be implemented using a loop that iterates over all sets or using a priority-queue based data structure. Preliminary testing showed that choosing one way or the other

greatly affects the speed comparison of the discussed heuristics. To avoid an implementation-dependent comparison, and because these aspects of the implementation are out of the scope of this work, we recorded the number of iterations instead.

Both heuristics are deterministic and only one run is required. The value of ϵ used in all DH runs is equal to $1 \times e^{-5}$. Smaller values of epsilon have caused numerical problems for some instances.

Our descent heuristic performed better than GH by finding better solutions for most of the test problems. For OR-Library benchmarks, DH found better solutions than GH for 47 instances, equal solutions for 10 instances, and worse solutions for 9 instances. For the airline, bus and railway scheduling problems, DH found better solutions than GH for all problems except one (equal solutions for RAIL516). The percentage deviations presented in figures 5.1, 5.2, 5.3 and 5.4 and the average percentage deviation presented in table 5.4 show that the solutions found by DH are also significantly better in quality than those found by GH (up to 7.41% better for OR-Library, up to 6.83% better for airline and bus problems, and up to 9.12% better for railway problems).

DH also performed fewer iterations than GH for most of the test problems. For OR-Library benchmarks, DH performed fewer iterations than GH for 56 instances, equal number of iterations for seven instances and more iterations for only two instances. For the airline, bus and railway scheduling problems, DH performed fewer iterations than GH for all problems except one (more iterations for BUS2). The average number of iterations performed by DH and GH is presented in table 5.4. The average number of iterations shows that the number of iterations performed by DH is significantly smaller than the number of iterations performed by GH. Thus, DH is theoretically faster than GH.

As a result, the proposed descent heuristic that is based on the new formulation performs better than the corresponding greedy heuristic that is based on the classical formulation by finding better results for most test problems using fewer iterations, which can lead to shorter computation times.

Table 5.1: OR-Library benchmarks

	Characteristics			Cost		Number of moves	
Instance	Size	Density	Best-known	DH	GH	DH	GH
4.1	200 × 1000	2%	429	433	434	77	93
4.2	200 × 1000	2%	512	523	552	75	94
4.3	200 × 1000	2%	516	531	546	79	96
4.4	200 × 1000	2%	494	503	507	70	93
4.5	200 × 1000	2%	512	515	518	72	95

Continued on next page

Table 5.1 – continued from previous page

	Characteristics			Cost		Number of moves	
Instance	Size	Density	Best-known	DH	GH	DH	GH
4.6	200 × 1000	2%	560	575	597	78	83
4.7	200 × 1000	2%	430	444	449	74	77
4.8	200 × 1000	2%	492	493	525	70	77
4.9	200 × 1000	2%	641	672	672	82	99
4.10	200 × 1000	2%	514	519	528	71	86
5.1	200 × 2000	2%	253	265	273	76	88
5.2	200 × 2000	2%	302	314	335	71	82
5.3	200 × 2000	2%	226	230	230	66	82
5.4	200 × 2000	2%	242	246	254	69	86
5.5	200 × 2000	2%	211	214	215	73	87
5.6	200 × 2000	2%	213	216	227	69	85
5.7	200 × 2000	2%	293	297	305	76	84
5.8	200 × 2000	2%	288	297	304	77	85
5.9	200 × 2000	2%	279	281	290	68	84
5.10	200 × 2000	2%	265	271	274	74	81
6.1	200 × 1000	5%	138	149	143	39	56
6.2	200 × 1000	5%	146	156	154	44	53
6.3	200 × 1000	5%	145	149	157	43	46
6.4	200 × 1000	5%	131	134	140	46	51
6.5	200 × 1000	5%	161	180	182	47	50
A.1	300 × 3000	2%	253	258	269	82	97
A.2	300 × 3000	2%	252	262	268	78	93
A.3	300 × 3000	2%	232	243	248	80	105
A.4	300 × 3000	2%	234	240	243	84	107
A.5	300 × 3000	2%	236	240	246	79	107
B.1	300 × 3000	5%	69	72	71	41	45
B.2	300 × 3000	5%	76	79	78	44	50
B.3	300 × 3000	5%	80	84	84	47	46
B.4	300 × 3000	5%	79	84	88	44	50
B.5	300 × 3000	5%	72	72	75	46	48
C.1	400 × 4000	2%	227	237	252	102	110
C.2	400 × 4000	2%	219	230	225	93	128
C.3	400 × 4000	2%	243	249	258	89	102
C.4	400 × 4000	2%	219	229	239	94	115
C.5	400 × 4000	2%	215	222	222	93	106
D.1	400 × 4000	5%	60	64	66	49	54
D.2	400 × 4000	5%	66	68	69	52	50
D.3	400 × 4000	5%	72	77	80	54	59
D.4	400 × 4000	5%	62	62	66	52	54
D.5	400 × 4000	5%	61	65	67	49	61

Continued on next page

Table 5.1 – continued from previous page

	Characteristics			Cost		Number of moves	
Instance	Size	Density	Best-known	DH	GH	DH	GH
E.1	500 × 5000	10%	29	30	30	30	35
E.2	500 × 5000	10%	30	33	35	31	37
E.3	500 × 5000	10%	27	29	31	29	31
E.4	500 × 5000	10%	28	32	31	32	33
E.5	500 × 5000	10%	28	30	30	30	32
F.1	500 × 5000	20%	14	16	17	16	17
F.2	500 × 5000	20%	15	16	16	15	16
F.3	500 × 5000	20%	14	17	15	17	17
F.4	500 × 5000	20%	14	17	15	17	14
F.5	500 × 5000	20%	13	15	15	17	15
G.1	1000 × 10000	2%	176	186	191	132	146
G.2	1000 × 10000	2%	154	166	176	115	139
G.3	1000 × 10000	2%	166	178	182	126	147
G.4	1000 × 10000	2%	168	178	179	128	138
G.5	1000 × 10000	2%	168	179	182	127	131
H.1	1000 × 10000	5%	63	69	69	68	65
H.2	1000 × 10000	5%	63	70	72	62	67
H.3	1000 × 10000	5%	59	63	66	62	62
H.4	1000 × 10000	5%	58	65	64	65	61
H.5	1000 × 10000	5%	55	60	61	61	60

Table 5.2 Airline and bus driver crew scheduling problems.

	Characteristics			Cost		Number of moves	
Instance	Size	Density	Best-known	DH	GH	DH	GH
AA03	106 × 8661	4.05%	33155	34637	35642	48	61
AA04	106 × 8002	4.05%	34573	36153	36749	45	62
AA05	105 × 7435	4.05%	31623	32249	32995	45	65
AA06	105 × 6951	4.11%	37464	38043	39422	43	70
AA11	271 × 4413	2.53%	35478	36965	39054	76	90
AA12	272 × 4208	2.52%	30815	33663	34044	77	85
AA13	265 × 4025	2.60%	33211	36337	37345	77	91
AA14	266 × 3868	2.50%	33219	36048	36530	77	95
AA15	267 × 3701	2.58%	34409	36269	37996	73	94
AA16	265 × 3558	2.63%	32752	36185	37160	79	85
AA17	264 × 3425	2.61%	31612	34326	36484	69	91
AA18	271 × 3314	2.55%	36782	39594	40603	84	101
AA19	263 × 3202	2.63%	32317	34749	36093	71	92
AA20	269 × 3095	2.58%	34912	37047	37744	82	86
BUS1	454 × 2241	1.89%	27947	28871	29673	88	100
BUS2	681 × 9524	0.51%	67760	69685	70606	282	280

Table 5.3 Railway crew scheduling problems.

	Characteristics			Cost		Number of moves	
Instance	Size	Density	Best-known	DH	GH	DH	GH
RAIL507	507 × 63009	1.2%	174	205	212	150	169
RAIL516	516 × 47311	1.3%	182	202	202	181	186
RAIL582	582 × 55515	1.2%	211	243	251	191	212
RAIL2586	2586 × 920683	0.4%	948	1102	1185	770	917
RAIL2536	2536 × 1081841	0.4%	691	828	891	581	660
RAIL4284	4284 × 1092610	0.2%	1065	1303	1385	997	1091
RAIL4872	4872 × 968672	0.2%	1534	1802	1900	1339	1521

Table 5.4 Average number of iterations and percentage deviations.

	Average number of iterations		Average percentage deviation	
Problems	DH	GH	DH	GH
OR-Library Benchmarks	64.89	74.51	5.46	7.31
Airline and bus problems	82.25	96.75	5.99	9.14
Railway problems	601.29	679.43	17.12	22.81

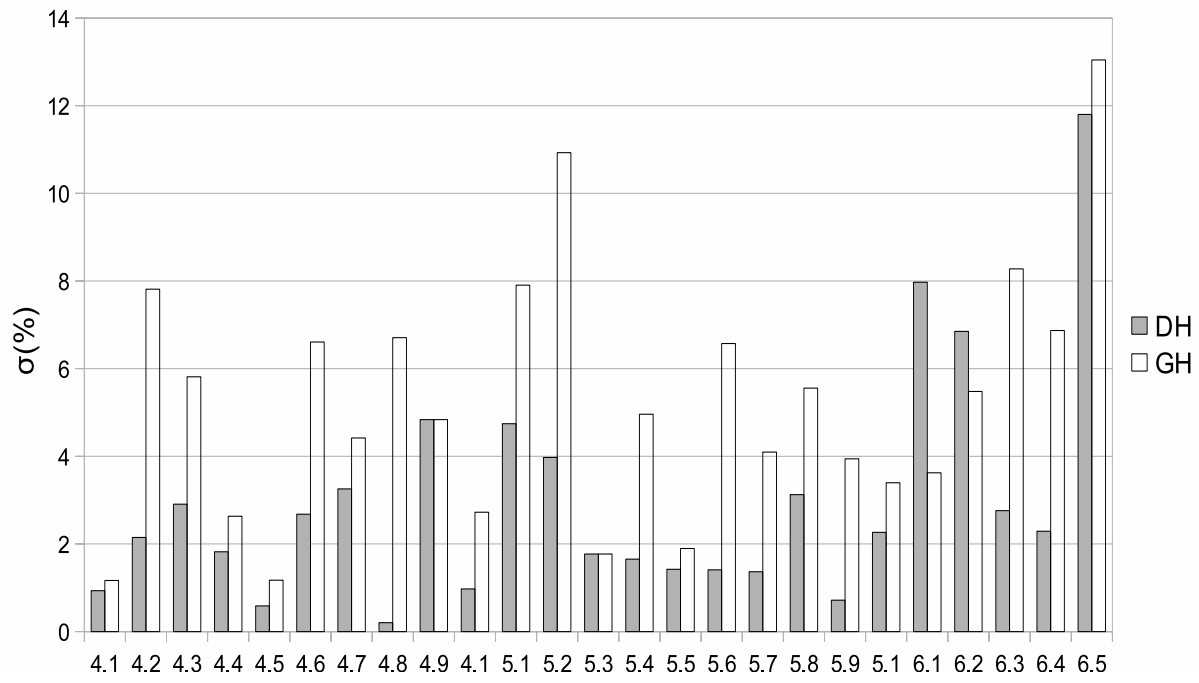

Figure 5.1 Percentage deviation from the best-known solution : OR-Library benchmarks 4.1 to 6.5.

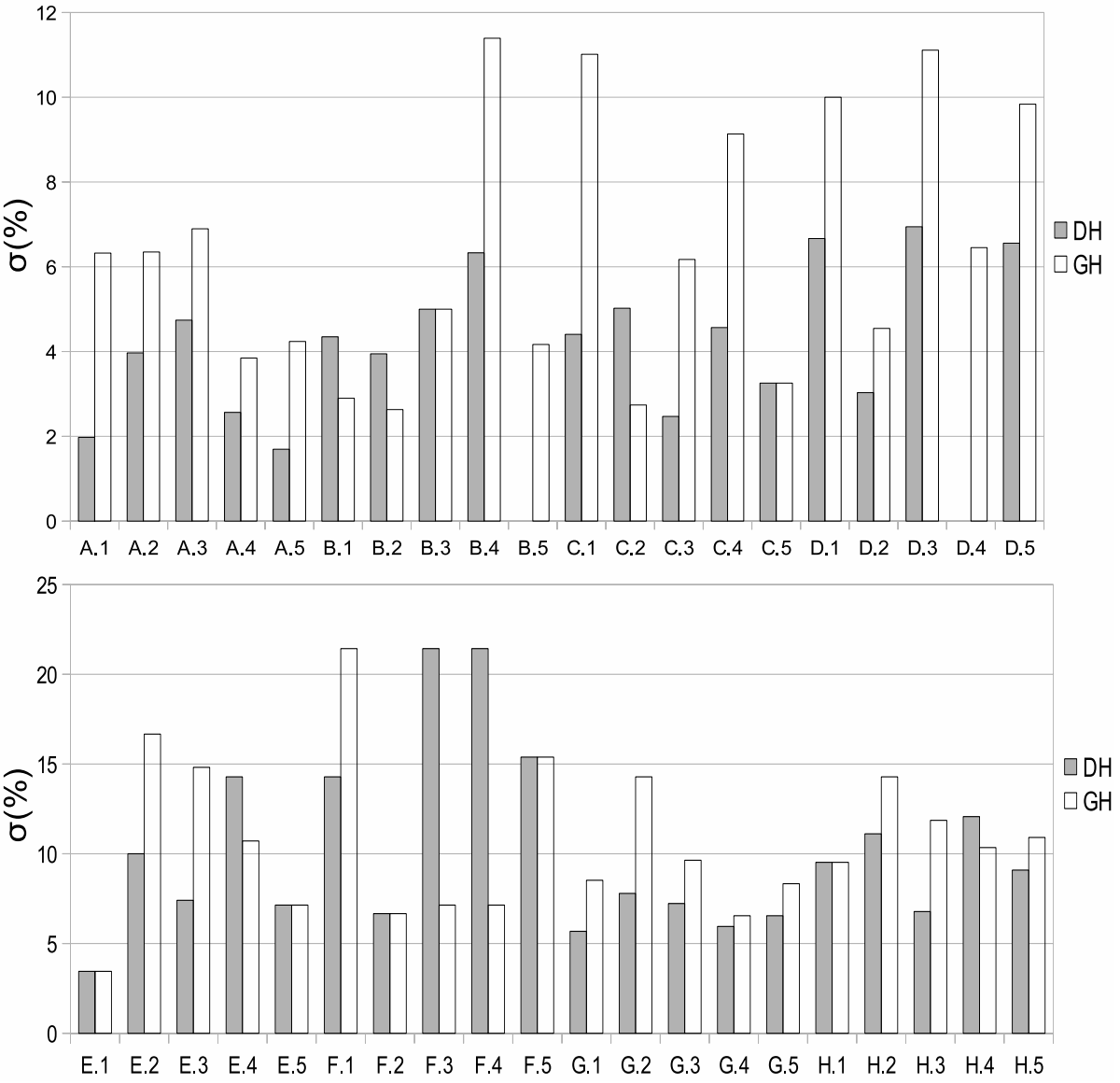

Figure 5.2 Percentage deviation from the best-known solution : OR-Library benchmarks A.1 to H.5.

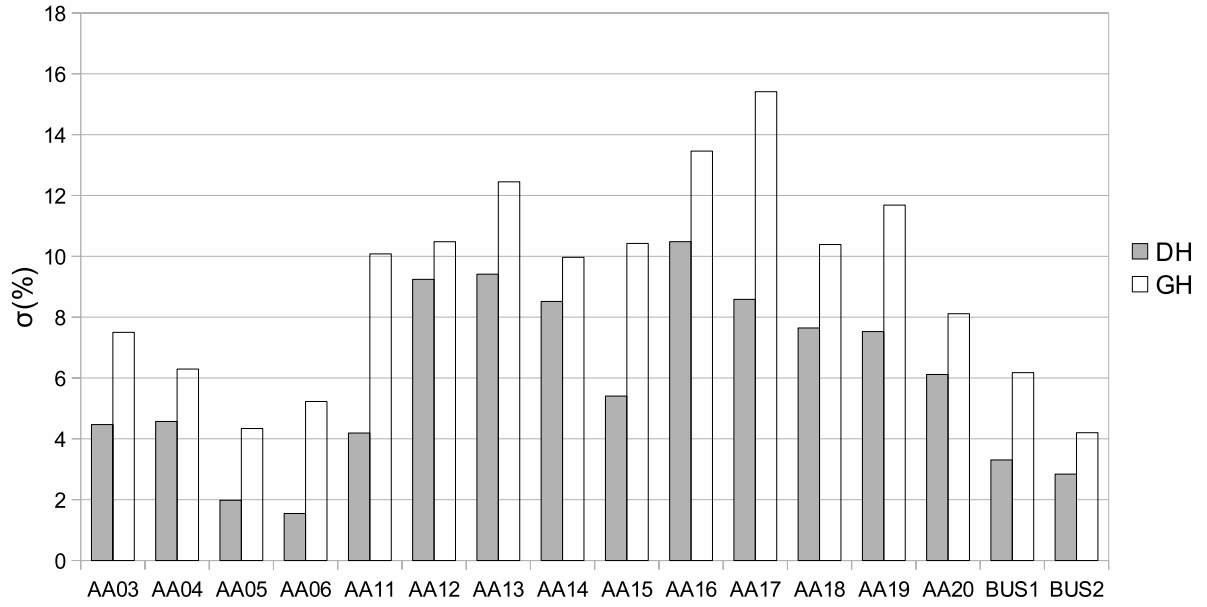

Figure 5.3 Percentage deviation from the best-known solution : Airline and bus scheduling problems.

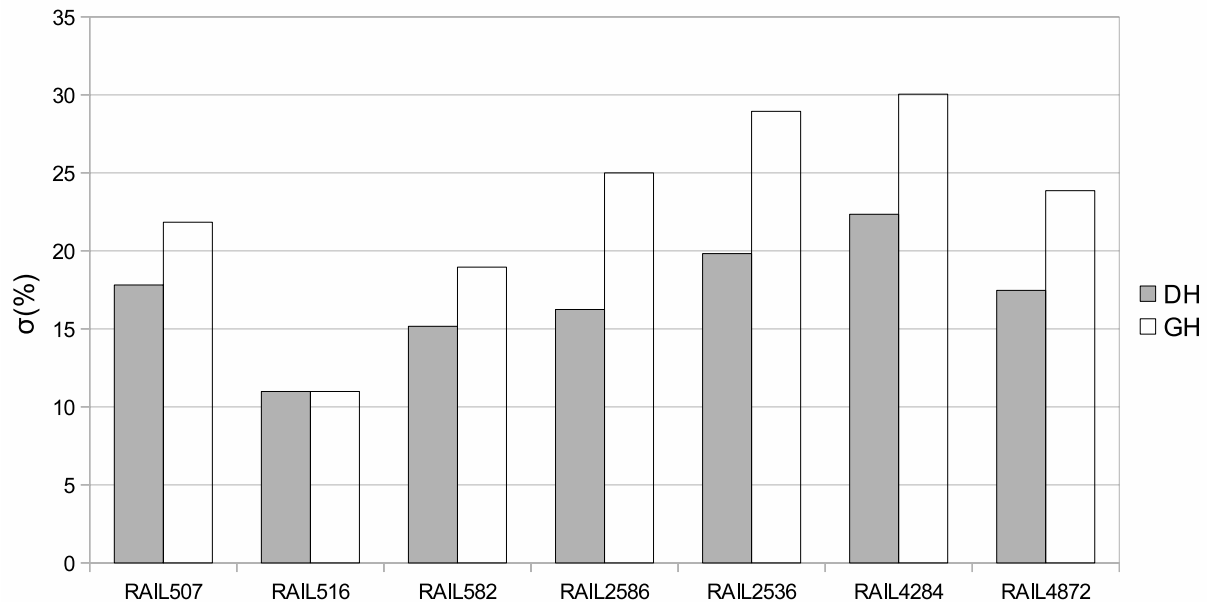

Figure 5.4 Percentage deviation from the best-known solution : Railway scheduling problems.

5.6 Conclusions and Future Work

In this paper, we identified two issues that arise when solving the SCP with metaheuristic approaches : solution infeasibility and set redundancy. We highlighted the difficulties of addressing these issues when solving the SCP with the different classes of metaheuristics, and proposed a new formulation that overcomes these difficulties. We showed that this formulation is, in fact, a new penalty approach that uses static penalty weights that are low enough to avoid disturbing the search but high enough to ensure the feasibility of the final solution. We also showed that all local optimums with respect to the new formulation and the 1-flip neighbourhood structure are feasible and free of redundant sets. As a result, building metaheuristic approaches for the SCP using the new formulation is straight forward.

To provide a first computational experience using the new formulation, we adapted a known greedy heuristic for the SCP to the new formulation and compared the adapted version to the original version using 88 set covering problems. The adapted version that is based on the new formulation found better solutions than the original version that is based on the classical formulation for 69 tests problems, equal solutions for ten problems, and worse solutions for nine problems. In addition, the adapted version performed fewer iterations than the original version for 78 test problems, equal number of iterations for two problems and more iterations for eight problems. Thus the adapted version finds better solutions than the original version in potentially shorter computation times. Moreover, the adapted version was easier to implement because we did not need to handle feasibility and set redundancy.

Most current metaheuristic approaches for the SCP incorporate a descent or greedy heuristic that is responsible for the intensification part of the search. Thus, having a more effective descent heuristic can lead to better metaheuristic approaches.

CHAPITRE 6

ARTICLE 3 : A GENETIC-TABU ALGORITHM FOR THE SET COVERING PROBLEM

Auteurs : Nehme Bilal, Philippe Galinier et Francois Guibault
Revue : Soumis à *Journal of Mathematical Modelling and Algorithms in Operations Research*

Abstract

In this paper, we present a hybrid metaheuristic that combines the genetic algorithm and tabu-search (GATS) to solve the set covering problem (SCP). GATS is a memetic algorithm in which a tabu-search (TS) heuristic is used as an improvement operator. In addition, the lengths of the tabu-lists of TS are adaptively adjusted by the genetic algorithm during mutation and crossover. The proposed algorithm has been tested on 65 test problems from OR-Library and has found the optimal solution to all of the problems in almost all the trials (647 of 650 trials). Additional testing has been performed on 23 airline, bus and railway scheduling problems and 22 unicost set covering problems. The computational results show that GATS performs better than most metaheuristic approaches for the SCP.

6.1 Introduction

The Set Covering Problem (SCP) is a popular optimization problem that has a wide range of industrial applications, including scheduling, manufacturing, service planning and location problems [22, 61, 6, 25]. The SCP is *NP-hard* in the strong sense [42]. The mathematical formulation of the SCP is as follows. Let $E = \{e_1, ..., e_m\}$ be a universe of elements and $S = \{s_1, ..., s_n\}$ be a collection of subsets $s_j \subseteq E$, where $\bigcup s_j = E$. Each set s_j covers at least one element of E and has an associated cost $c_j > 0$. The objective is to find a subcollection of sets $X \subseteq S$ that covers all of the elements in E at a minimal cost. The mathematical programming model of the SCP is usually formulated as follows. Let

— $A^{m \times n}$ be a zero-one matrix where $a_{ij} = 1$ if element i is covered by set j, and $a_{ij} = 0$ otherwise.

— $X = \{x_1, x_2, ..., x_n\}$ where $x_j = 1$ if set j (with cost $c_j > 0$) is part of the solution, and $x_j = 0$ otherwise.

Minimize

$$\sum_{j=1}^{n} c_j x_j \tag{6.1}$$

Subject to

$$1 \leq \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, ..., m \tag{6.2}$$

$$x_j \in \{0, 1\} \quad i = 1, ..., n \tag{6.3}$$

If all the sets have equal costs, the problem is called the unicost set covering problem (USCP).

In this paper, we present a hybrid metaheuristic to solve the SCP. The proposed algorithm is a memetic algorithm that combines the genetic algorithm and tabu-search and is called the genetic-tabu algorithm (GATS). In GATS, each mated individual of the GA is improved using a short run of tabu-search. In addition, we propose a new and simple mechanism of adaptively adjusting the tabu-list lengths of TS using the mutation and crossover operators of the GA.

GATS is compared to the most popular metaheuristic approaches for the SCP on 65 test problems from OR-Libray [15]. OR-Library problems have been used in the literature to evaluate most metaheuristic approaches for the SCP. In addition, GATS is tested on 22 USCP problems and 23 real-world SCP problems that contain very difficult and large-scale problems.

The paper is organized as follows. In the next section, a literature review of existing algorithms for the SCP, emphasizing metaheuristic approaches, is presented. The GATS algorithm is presented in section 6.3 and a computational analysis is presented in section 6.4. The paper ends with a summary (section 6.5).

6.2 Literature Review

The SCP has been solved to optimality or near optimality using both exact and heuristic algorithms.

6.2.1 Exact methods

Most exact algorithms for the SCP are based on linear integer programming approaches, especially branch-and-bound methods. In these methods, a lower bound is computed at each node of the branching tree using a linear or Lagrangian relaxation. Among the exact methods, a branch-and-bound algorithm based on a dual heuristic was used to solve SCP problems

with up to 200 elements and 2000 sets [38]. Beasley et Al. [17] solved larger problems (up to 400 elements and 4000 sets) by improving the branching strategy of the branch-and-bound algorithm and combining it with a Lagrangian heuristic and Gomory f-cuts. Harche et Al. [48] developed a new branch-and-bound algorithm called *column substraction*, which is capable of solving sparse problems with up to 800 elements and 9000 sets.

6.2.2 Heuristic methods

Although exact methods can theoretically find the optimal solution to any SCP problem, the computation times involved in solving large-scale problems to optimality can be very long. In such cases, heuristic algorithms are used to obtain good, near-optimal, or even optimal solutions in reasonable computation times. Several approaches developing heuristic algorithms for the SCP have been investigated in the literature. Among these approaches are heuristic algorithms based on Lagrangian relaxations [14, 26, 23, 92, 7, 21], heuristics related to primal-dual methods [18, 69, 93] and metaheuristic approaches, such as the genetic algorithm [16, 35, 84, 2] ant colony optimization [63, 78, 80], local search [72, 92], Meta-RaPS [61] and electromagnetic metaheuristics [73].

A state-of-the-art algorithm for the SCP based on Lagrangian relaxation was developed in [22]. CFT is able to find the optimal solution to all non-unicost SCP instances of OR-Library in all trials with a relatively short computation time. Another very effective approach for the non-unicost SCP is 3-FNLS [92]. 3-FNLS is a hybrid heuristic that combines a 3-flip local-search algorithm and Lagrangian relaxation. 3-FNLS performs particularly well on large-scale problems.

6.2.3 Metaheuristic approaches

Metaheuristic approaches perform very well on the SCP but do not match the performance of Lagrangian-based approaches. Moreover, metaheuristic approaches have not been tested on the large-scale problems (to our knowledge) that have been used to test Lagrangian-based approaches.

Most metaheuristic algorithms for the SCP have only been tested on OR-Library [15]. These algorithms can find the optimal solutions to most OR-Library problems but have relatively low hit frequencies ; the *hit frequency* is the ratio of the number of trials in which the optimal (or best-known) solution is found to the total number of trials. Among these approaches, a genetic algorithm (BCGA) for the non-unicost SCP is presented in [16]. The main characteristics of BCGA are : a new fitness-based crossover operator for binary string genomes called the fusion crossover, a variable mutation rate, a greedy-like repair heuristic,

and a redundancy removal heuristic that improves the quality of the repaired solutions. On one hand, BCGA is able to find the optimal solutions to 61 of the 65 non-unicost SCP problems of OR-Library. On the other hand, the hit frequencies of BCGA are less than 0.5 for 12 problems and between 0.5 and 0.9 for 32 problems (the hit frequencies of the metaheuristics discussed in this section are presented in Table 6.3). Eremeev [35] developed a genetic algorithm with a non-binary solution-representation (NBGA). An individual is represented by a string of integers where each gene g_i is equal to j if the element i is covered by the set j. Because each gene g_i is always associated with a set j, all the elements are covered and all solutions are feasible. As in BCGA, a redundancy removal operator is used to enhance the quality of the solutions. NBGA has found the optimal solutions to 63 of the 65 non-unicost problems of OR-Library. The hit frequencies of NBGA are less than 0.5 for 5 problems and less than 1.0 for up to 38 problems. Aickelin [2] proposed an indirect genetic algorithm (IGA) that addresses the feasibility constraint by decoupling the actual solution space from the search space of the genetic algorithm. The genetic algorithm no longer optimizes the actual problem directly ; instead, it tries to find an optimal ordering of all elements to be covered, which is then exploited by a secondary decoding routine. IGA has also been tested on the same 65 non-unicost SCP problems of OR-Library. IGA has found the optimal solution to 62 of the 65 problems. Unfortunately, the computational results reported are not sufficient to calculate the hit frequencies. The limited information shows that the hit frequencies are less than 1.0 for up to 25 problems. A recent and popular metaheuristic for the SCP is Meta-RaPS [61]. The authors of Meta-RaPS report experimental results containing the optimal solution to all of the 65 non-unicost SCP problems of OR-Library. However, the detailed results available in Lan's thesis [60] show that the hit frequencies are less than 0.5 for 55 problems and less than 1.0 for all problems. Moreover, the hit frequencies are as low as 0.002 for large problems and as low as 0.09 for small problems. These low hit frequencies seem to indicate that Meta-RaPS does not perform very well on the SCP. Another recent metaheuristic approach, based on electromagnetism theory (EM), is presented in [73]. After generating a pool of solutions (the initial population), a fixed number of local search iterations and electromagnetic movements are applied. The algorithm also incorporates a mutation operator to escape from local optima. EM finds the optimal solution to 53 of the 65 non-unicost problems of OR-Library. The number of trials performed and the hit frequencies are not reported. Meta-RaPS and EM have also been tested on the USCP, where they show a similar performance.

One of the best metaheuristic approaches for the USCP is the local search algorithm presented in [72]. The proposed algorithm uses a new method to generate the neighborhood of a solution based on the number of sets in the current best solution. For a given solution, all

the neighbors that have more sets than the current best solution are not allowed. It also uses a tabu-list to avoid cycling. Unfortunately, the proposed algorithm is specifically designed for the USCP and cannot be used to solve non-unicost problems.

6.3 Proposed Genetic-Tabu Algorithm

The proposed genetic-tabu algorithm is a hybrid metaheuristic that combines the genetic algorithm (GA) and tabu-search (TS). The genetic algorithm is an evolutionary algorithm that uses the concepts of natural selection, recombination and mutation to evolve a population of solutions. Tabu-search (TS) is a local search algorithm that uses a search history to escape from local optima and cycles.

Evolutionary algorithms tend to optimize globally, whereas local search algorithms tend to optimize locally. For these reasons, evolutionary algorithms are generally powerful in diversifying the search but weak in intensifying it, and local search algorithms are generally powerful in intensifying the search but week in diversifying it. Thus, evolutionary and local search algorithms provide complementary strengths and weaknesses that motivate the combination of the two.

Tabu-search and the genetic algorithm have been frequently combined in the literature. The most common form of a GA and TS hybrid uses tabu-search to improve the quality of the solutions at the end of each generation [89, 88, 39]. Similarly, in [96, 50], tabu-search is used to optimize the solutions at the beginning of each generation. Other proposed hybrids use tabu-search to replace the mutation operator [30] or the selection operator [51, 59]. Tabu-search has also been successfully combined to other metaheuristics such as iterated-local-search [20] and scatter-search [45].

An important difficulty encountered when designing a good tabu-search algorithm is choosing the length of the tabu-list. In fact, the length of the tabu-list can significantly affect the performance of tabu-search. A fixed length of *seven* is often used in the literature but better results are generally found by adaptively adjusting the tabu-list length. In most adaptive adjustment approaches, the length of the tabu-list is adjusted by exploiting the search history : the length of the list is increased when a high rate of cycling is detected in the search history and decreased when cycling is less frequent. Among the adaptive tabu-search approaches are *reactive tabu search* [12], the *cancellation sequence method* [32, 43] and the *reverse elimination method* [43, 33]. These approaches share the characteristic of being sophisticated and difficult to implement.

In GATS, at the end of each generation of the GA, the individuals are improved using TS. In addition, we use the GA to adaptively adjust the length of the tabu-lists used in TS. Each

individual holds its own tabu-list lengths as part of its genotype. During the evolution of the GA, the tabu-list lengths are exchanged between individuals during crossover and randomly modified during mutation. The tabu-list lengths that produce good individuals are more likely to survive in the population than the ones that produce poor individuals. Similarly, the tabu-list lengths inherited from fit parents are more likely to produce good children.

6.3.1 SCP formulation

The feasibility constraints of the SCP makes designing a tabu-search algorithm for the SCP a difficult task. To overcome this difficulty, we use a new formulation of the SCP that naturally penalizes infeasible solutions and redundant sets, which makes designing a tabu-search algorithm for the SCP an easier task. In addition, this formulation eliminates the need to use a repair or redundancy removal heuristic in the GA. The new formulation has been introduced, analyzed and tested in [19] and is as follows.

Let

— $A^{m \times n}$ be a zero-one matrix where $a_{ij} = 1$ if element e_i is covered by set j and $a_{ij} = 0$ otherwise.

— $X = \{x_1, x_2, ..., x_n\}$ where $x_j = 1$ if set s_j (with cost $c_j > 0$) is part of the solution and $x_j = 0$ otherwise.

— $Y = \{y_1, y_2, ..., y_m\}$ where $y_i = 1$ if element e_i (with gain $g_i > 0$) is covered in the solution and $y_i = 0$ otherwise.

Maximize

$$\sum_{i=1}^{m} g_i y_i - \sum_{j=1}^{n} c_j x_j \tag{6.4}$$

subject to

$$y_i \leq \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, ..., m \tag{6.5}$$

$$x_j, y_i \in \{0, 1\} \quad j = 1, ..., n \tag{6.6}$$

Let $c_{min}(e_i)$ be the cost of the cheapest set among the sets that cover the element e_i. A gain $g_i = c_{min}(e_i) + \epsilon$ is attributed to each element e_i where ϵ is a small positive constant.

Constraint (6.5) is a relaxation of constraint (6.2) because it does not impose coverage of all the elements ; its only purpose is to keep track of which elements of E are covered.

It is shown in [19] that this formulation has the same optimal solution as the classical SCP formulation. In addition, it is shown that all local optima with respect to the new formulation and a 1-flip neighborhood structure are feasible and free of redundant sets.

6.3.2 Solution representation

A binary string solution representation is used in GATS where the ith bit is equal to one if the set i is part of the solution and 0 otherwise.

Let X be a given configuration (a subset of S that represents a feasible or infeasible solution), S_X be the sets used in X and E_X be the elements covered by X. The objective score of X is equal to the following (which is equivalent to the objective function (6.4)) :

$$\text{score}(X) = \sum_{e_i \in E_X} g_i - \sum_{s_j \in S_X} s_j \tag{6.7}$$

6.3.3 Tabu-search (TS)

The tabu-search algorithm (TS) is a steepest descent algorithm that uses a simple 1-flip neighborhood structure and two tabu-lists. In each iteration, the current configuration is replaced with its best non-tabu neighbor, the one with the highest objective score. In addition, a tabu move can be performed if it satisfies a predefined condition, called the aspiration criterion. Our aspiration criterion allows a tabu move to be performed if the resulting solution is better than all of the solutions that have been previously visited.

A neighbor X' of a given configuration X can be obtained by performing an *add* or *remove* move. *add(j)* is the move that adds the set s_j to the configuration (flips x_j from 0 to 1), and *remove(j)* is the move that removes the set s_j from the configuration (flips x_j from 1 to 0). Let δ_j be the variation of the objective score associated with adding (removing) the set s_j to (from) the configuration. δ_j can be calculated as follows :

$$\delta_j = \begin{cases} \sum_{i=1}^{m} g_i(1 - y_i) - c_j & \text{if}(x_j = 0) \\ c_j - \sum_{j=1}^{m} g_i(1 - y_i) & \text{if}(x_j = 1) \end{cases}$$

where c_j is the cost of set s_j, g_i is the gain of element e_i and $y_i = 1$ if e_i is covered ($y_i = 0$ otherwise). In each iteration of TS, the non-tabu move that is associated with the highest value of δ_j or a tabu move that satisfies the aspiration criterion, is performed. In addition, we use a separate tabu-list for each type of move (*add* and *remove*) and the lengths of the tabu-lists are not necessarily equal. TS is stopped if the best solution is not improved after a given number of iterations (*nStag*). The pseudo-code of TS is presented in Algorithm 7. The procedure *find-next-move* returns the best non-tabu move or a tabu move that passes the aspiration criterion.

Algorithm 7 TS(X)

```
clear-tabu-lists() ;
best ← X ;
stagnationCounter ← 0 ;
loop
   m ← find-next-move(X) ;
   if (x_m = 0) then
      x_m ← 1 ;
   else
      x_m ← 0
   end if
   mark-tabu(m) ;
   if (score(X) > score(best)) then
      best ← X ;
      stagnationCounter ← 0 ;
   else if (stagnationCounter > nStag) then
      return ;
   end if
   stagnationCounter ← stagnationCounter + 1 ;
end loop
```

6.3.4 The Genetic algorithm

An incremental genetic algorithm is used in GATS in which only two children are created at the end of each generation. In addition to the binary string solution representation, each individual holds two integers that are the lengths of the tabu-lists associated with the *add* and *remove* moves. These values are used by the TS operator during the improvement phase of GATS. Similarly to the other genes of an individual, the lengths of the tabu-lists are subject to mutation and crossover. The operators of GATS are described as follows.

The initialization operator initializes each individual by randomly assigning a value to each bit of the corresponding binary string. The tabu-lists lengths associated with each individual are also randomly initialized by assigning a value between zero and a maximum length (user defined) to each tabu-list length.

A unidirectional bit flip mutation operator is used in GATS (drop mutation). The drop mutation operator flips each bit of a given individual from 1 to 0 according to a given mutation probability. This unidirectional bit flip mutation appears to be better than the classical bidirectional flip mutation because it avoids adding too many redundant sets to the solutions. In fact, if too many redundant sets are added during mutation, TS will spend excessive time removing redundant sets, which will slow down GATS. The tabu-list lengths are also mutated using the same mutation probability, where a random value between 0 and the maximum length is attributed to each tabu-list length.

The selection operator uses the roulette-wheel scheme, where the probability of selecting

a given individual is proportional to its objective score (calculated using Equation 6.7).

For crossover, the uniform crossover scheme, where two parents are combined to produce two children, is used. In the uniform crossover scheme, each gene of the generated children has a 50% chance to be chosen from one parent or the other. The tabu-list lengths are also crossed-over using the same uniform crossover scheme. Thus, the tabu-list lengths of a given child can be both inherited from the same parent or each from a different parent.

6.4 Computational Results

We tested GATS on four classes of problems :

— **OR-Library benchmarks :** This class includes 65 small and medium sized randomly generated problems that were frequently used in the literature of the SCP. These problems are available in OR-Library [15].

— **Airline and bus scheduling problems :** This class includes fourteen real world airline scheduling problems (AA instances) and two bus driver scheduling problems (BUS instances). These problems were obtained from [93].

— **Railway scheduling problems :** This class includes seven large-scale railway crew scheduling problems from Italian railways and are available in OR-Library [15].

— **Unicost Problems :** This class includes 22 combinatorial optimization problems modeled as unicost set covering problems. Unicost problems are generally solved with specialized algorithms that take advantage of the unicost property. These problems were gathered from [15] and [93].

The characteristics of all test problems are presented in Table 6.1.

GATS is compared to the most popular metaheuristic approaches for the SCP on OR-Library benchmarks because they are the only non-unicost SCP problems that have been solved with metaheuristic approaches. In addition, because most of the OR-Library benchmarks are relatively small for modern computers (except for G and H problems), we tested GATS on larger and more difficult problems that have been solved with Lagrangian-based approaches. These problems are airline, bus and railway scheduling problems. We have also tested GATS on unicost problems to investigate the possibility of using or adapting GATS for the USCP.

The parameters used are similar for all test problems. We used a population of fifty individuals, a mutation rate of 0.3 and a crossover probability of 0.9. The maximum length of the tabu-list is 200 (for both tabu-lists). When improving an individual, the tabu-search operator is stopped after 100 non-improving iterations ($nStag$ in Algorithm 7). For the railway scheduling problems, we used $nStag = 1000$ because these problems are larger and requires

Table 6.1 Test problems characteristics

Problems	Number of elements	Number of sets	Density
OR-Library benchmarks			
4	200	1000	2%
5	200	1000	2%
6	200	1000	5%
A	300	3000	2%
B	300	3000	5%
C	400	4000	2%
D	400	4000	5%
E	500	5000	10%
F	500	5000	20%
G	1000	10000	2%
H	1000	10000	5%
Unicost Problems			
E1	50	500	20%
E2	50	500	20%
E3	50	500	20%
E4	50	500	20%
E5	50	500	20%
CLR10	511	210	12.3%
CLR11	1023	330	12.4%
CLR12	2047	495	12.5%
CLR13	4095	715	12.5%
CYC06	240	192	2.1%
CYC07	672	448	0.9%
CYC08	1792	1024	0.4%
CYC09	4608	2304	0.2%
CYC10	11520	5120	0.8%
CYC11	28160	11264	0.02%
STS9	12	9	33.3%
STS15	35	15	20%
STS27	117	27	11.1%
STS45	330	45	6.7%
STS81	1080	81	3.7%
STS135	3015	135	2.2%
STS243	9801	243	1.2%
Airline and bus scheduling problems			
AA03	106	8661	4.05%
AA04	106	8002	4.05%
AA05	105	7435	4.05%
AA06	105	6951	4.11%
AA11	271	4413	2.53%
AA12	272	4208	2.52%
AA13	265	4025	2.60%
AA14	266	3868	2.50%
AA15	267	3701	2.58%
AA16	265	3558	2.63%
AA17	264	3425	2.61%
AA18	271	3314	2.55%
AA19	263	3202	2.63%
AA20	269	3095	2.58%
BUS1	454	2241	1.89%
BUS2	681	9524	0.51%
Railway scheduling problems			
RAIL507	507	63009	1.2%
RAIL516	516	47311	1.3%
RAIL582	582	55515	1.2%
RAIL2536	2536	1,081841	0.4%
RAIL2586	2586	920683	0.4%
RAIL4284	4284	1,092610	0.2%
RAIL4872	4872	968672	0.2%

The density is the percentage of "ones" in the constraints matrix of the SCP (see Equation 6.2).

longer runs of tabu-search. GATS was implemented in C++ and compiled using the Intel C++ Compiler. The tests were performed on an Intel Westmere 2.67 GHz (using a single processor). Ten trials were performed on each test problem. The maximum duration of each trial was two hours for all problems except for the railway scheduling problems where a maximum duration of 24 hours was used.

6.4.1 OR-Library benchmarks

The detailed results of the experiments performed on OR-Library benchmarks are presented in Table 6.2, where the hit frequency is the ratio of the number of times the best-known solution was found over the total number of trials and the solution-time is the time when the best solution was found for the first time. Ten trials were performed on each test problem. The problems in each of the classes 4, 5, 6, 7, A, B, C, D, E, and F are similar to each other and have been averaged in the first rows of the table. The obtained results show that GATS was able to find the optimal solution to all the test problems in almost all the trials (647 of 650 trials). The average solution times are also reasonably short (0-80 seconds).

In Table 6.3, GATS is compared to four of the most popular metaheuristic approaches for the SCP. These algorithms are Beasley and Chu's genetic algorithm (BC) [16], a genetic algorithm with a non-binary representation (NBGA) [35], an indirect genetic algorithm (IGA) [2] and Meta-RaPS [61]. Because BC, NBGA and IGA were executed on older hardware, it is difficult to compare the computation times of GATS and Meta-RaPS with these algorithms. We present the original solution times of all the algorithms and their corresponding hardware information instead of resorting to inaccurate converted times. All of the algorithms perform ten trials on each problem, except Meta-RaPS where the number of trials is unknown but seems high (based on the detailed results of Meta-RaPS presented in Lan's thesis [60]).

The obtained hit frequencies show that GATS and Meta-RaPS are the only algorithms that find the optimal solutions to all problems. In addition, the hit frequencies show that GATS finds the optimal solutions in 99.5% of its trials, whereas BC, NBGA, IGA and Meta-RaPS find the optimal solution in 63%, 75%, (62 to 86)% and 17% of their trials, respectively. On the other hand, the hit frequencies are usually dependent on the computation times, which are difficult to compare in our case (as discussed earlier). As a result, this comparison is only an approximation (which is also the case for most comparisons found in similar studies). The overall average solution times in seconds are 23.51 for GATS, 226.58 for BC, 141.37 for NBGA, 231.48 for IGA and 375.87 for Meta-RaPS. The obtained computation times show that GATS is faster than Meta-RaPS in most cases and that Meta-RaPS is likely slower than BC, NBGA and IGA (because Meta-RaPS has longer computation times on newer hardware). Although the obtained computation times of GATS are also shorter than those

of BC, NBGA and IGA, GATS is not necessarily faster (and is possibly slower) than these algorithms because GATS is executed on significantly faster hardware.

Table 6.2 Detailed results for OR-Library benchmarks

Problems	BKS	% deviation from BKS min	max	avg	SD	HF	T_{avg}(s)	Generations	Generations/s
4	-	0	0	0	0	1	0.01	4420297.4	613.93
5	-	0	0	0	0	1	0.9	2383204.7	331
6	-	0	0	0	0	1	0	4188526.0	581.74
A	-	0	0	0	0	1	2.18	1537976.6	213.61
B	-	0	0	0	0	1	2	1439320.3	199.91
C	-	0	0	0	0	1	3.04	1133214.1	157.39
D	-	0	0	0	0	1	3.07	1044050.5	145.01
E	-	0	0	0	0	1	6.38	657275.4	91.29
F	-	0	0	0	0	1	8.6	393958.2	54.72
G.1	176	0	0	0	0	1	22.11	409811.1	56.9
G.2	154	0	0.65	0.12	0.4	0.8	22.22	406874.2	56.5
G.3	166	0	0.6	0.05	0.3	0.9	28.78	391805.1	54.4
G.4	168	0	0	0	0	1	24.56	395543.9	54.9
G.5	168	0	0	0	0	1	22.11	378188.3	52.5
H.1	63	0	0	0	0	1	80	332900.0	46.2
H.2	63	0	0	0	0	1	25.78	305391.7	42.4
H.3	59	0	0	0	0	1	28.11	328218.9	45.6
H.4	58	0	0	0	0	1	30.22	283815.0	39.4
H.5	55	0	0	0	0	1	22.67	311644.8	43.3

BKS is the best-known solution value before GATS (these solutions are optimal for OR-Library instances), $T_{avg}(s)$ is the average solution-time in seconds, *SD* is the standard deviation of the solutions found in the all trials, *HF* is the hit frequency, *generations* is the average of the total number of generations performed in all trials, and *generations/s* is the average number of generations per second.

6.4.2 Airline and bus scheduling problems

The airline and bus scheduling problems are significantly harder to solve than OR-Library benchmarks. These problems were solved by Lagrangian-based approaches such as [7], [21] and [92].

The detailed results for these problems are presented in Table 6.4. Ten trials were performed on each test problem. The obtained results show that GATS performs very well on airline and bus scheduling problems by finding the best-known solution for thirteen of the sixteen test problems. For the problems where GATS did not find the best-known solution, the minimum percentage deviation from the best-known solution varies between 0.13% and 0.37%.

Because the airline and bus scheduling problems have not been solved with other metaheuristic-only approaches (to our knowledge), there are no other algorithms to which we can compare our results. Instead, we experiment with a variant of GATS (denoted GAGr) in which the

Table 6.3 Comparison of GATS with other metaheuristics on OR-Library benchmarks

	Hit frequencies					Average solution-times (s)				
Problems	GATS	BC	NBGA	IGA	Meta-RaPS	GATS Intel 2.67G	BC R400 100M	NBGA Intel 100M	IGA Intel 450M	Meta-RaPS Intel 1.8G
4	1	0.88	0.86	1	0.35	0.01	35.96	35.3	93.3	0.83
5	1	0.76	0.7	1	0.44	0.9	21.77	40.9	61.2	0.58
6	1	0.94	0.86	1	0.29	0	20.18	40.05	7.6	0.33
A	1	0.86	0.44	1	0.19	2.18	52.78	82	81	6.57
B	1	1	1	1	0.31	2	54.9	20.8	30.4	0.72
C	1	0.68	0.74	1	0.09	3.04	70.34	53.5	82.8	13.13
D	1	0.96	0.94	0.1-0.9	0.24	3.07	81.36	26.62	69	3.7
E	1	0.74	1	1	0.3	6.38	99.66	26.4	56	18.66
F	1	0.86	0.86	1	0.2	8.6	55.34	116.48	142.8	9.73
G1	1	0.2	0.7	0.1-0.9	0.036	22.11	361	115	144	298.97
G2	0.8	0	0.5	0	0.002	22.22	127.4	318.3	327	222.34
G3	0.9	0.1	0.1	0.1-0.9	0.004	28.78	249.7	627.6	408	21.56
G4	1	0.2	0.4	0.1-0.9	0.002	24.56	532.3	160	303	194.21
G5	1	0.2	0.7	0.1-0.9	0.040	22.11	194.1	161.2	532	47.57
H1	1	0	0	0.1-0.9	0.004	80	594.4	90.5	668	3917.08
H2	1	0	0	0	0.004	25.78	187.4	34.7	443	238.45
H3	1	0.9	1	0.1-0.9	0.002	28.11	637.3	493.2	648	783.2
H4	1	0.4	1	0	0.018	30.22	770.3	218.2	235	1358.28
H5	1	0.9	1	0.1-0.9	0.076	22.67	158.9	25.2	66	5.62

BC refers to [16], NBGA refers to [35], IGA refers to [2] and Meta-RaPS refers to [61].

TS operator is replaced with a greedy heuristic. The greedy heuristic is similar to the TS operator with the differences that it does not use a search history (i.e. no tabu-list) and that it stops when a local optimum is reached. Computational results presented in Table 6.5 show that GAGr performs worse than GATS (GAGr finds the best-known solution to only ten of the sixteen test problems) ; which shows that TS makes GATS more effective. On the other hand, the differences between the solutions found by GAGr and those found by GATS are too small to be conclusive. In the next section, GATS and GAGr are compared on larger and more difficult problems.

6.4.3 Railway scheduling problems

Similarly to the airline and bus scheduling problems, the railway scheduling problems have only been solved using Lagrangian-based heuristics [7, 21, 92, 26]. These problems are the largest among all the problems discussed in this paper. The detailed results of GATS are presented in Table 6.6.

The obtained results show that GATS is not able to find the best-know solution for any of the railway scheduling problems. We notice that the deviations from the best-known solutions are small for the first three problems but increase significantly with the problem's size. From

Table 6.4 Detailed results for the airline and bus scheduling problems

Problems	BKS	% deviation from BKS min	max	avg	SD	HF	T_{avg}(s)	Generations	Generations/s
AA03	33155	0	0	0	0	1	24.5	543469.6	75.5
AA04	34573	0	0	0	0	1	7.5	617487.6	85.8
AA05	31623	0	0	0	0	1	0.9	640754.3	89.0
AA06	37464	0	0	0	0	1	312.4	679499.6	94.4
AA11	35384	0	0.51	0.41	58.69	0.1	477.6	980044.6	136.1
AA12	30809	0	0.05	0.04	6.75	0.2	116.8	1072980.3	149.0
AA13	33211	0	0.3	0.06	34.82	0.6	105.7	1102165.0	153.1
AA14	33219	0	0.25	0.09	37.77	0.6	2392.4	1114375.1	154.8
AA15	34409	0	0.22	0.05	31.62	0.7	1507.4	1202049.3	167.0
AA16	32752	0	0.34	0.24	34.33	0.1	2257.5	1256256.4	174.5
AA17	31612	0	0	0	0	1	15	1253159.6	174.0
AA18	36782	0	0.06	0.03	6.2	0.2	850.1	1305934.1	181.4
AA19	32317	0.13	0.66	0.35	65.44	0	2251.3	1351291.4	187.7
AA20	34912	0.18	0.18	0.18	0	0	44.7	1444394.4	200.6
BUS1	27947	0	0.1	0.07	11.8	0.2	4026.56	4492131.6	623.9
BUS2	67760	0.37	0.53	0.44	39.68	0	6642.78	858985.9	119.3

This table is similar in format to Table 6.2.

Table 6.5 Comparison of GATS and GAGr on the airline and bus scheduling problems

Problems	BKS	% deviation from BKS GATS min	GATS avg	GAGr min	GAGr avg	T_{avg}(s) GATS	GAGr
AA03	33155	0	0	0	0	24.5	3895
AA04	34573	0	0	0	0	7.5	0.2
AA05	31623	0	0	0	0	0.9	0
AA06	37464	0	0	0	0	312.4	430.4
AA11	35384	0	0.41	0	0.33	477.6	4171.8
AA12	30809	0	0.04	0.05	0.05	116.8	1.4
AA13	33211	0	0.06	0	0.02	105.7	1754
AA14	33219	0	0.09	0	0.34	2392.4	9673.4
AA15	34409	0	0.05	0	0.04	1507.4	2350.4
AA16	32752	0	0.24	0.05	0.24	2257.5	4701
AA17	31612	0	0	0	0	15	615.6
AA18	36782	0	0.03	0.03	0.07	850.1	2436.4
AA19	32317	0.13	0.35	0.13	0.13	2251.3	3788.8
AA20	34912	0.18	0.18	0.18	0.18	44.7	5.6
BUS1	27947	0	0.07	0	0	4026.56	9496.4
BUS2	67760	0.37	0.44	0.38	0.44	6642.78	6169.2

BKS is the best-known solution value before GATS and $T_{avg}(s)$ is the average solution-time in seconds.

Table 6.6, we observe that the average number of generations performed by GATS in 24 hours is low (as low as 628.5) ; which shows that GATS is slow on large-scale problems.

In Table 6.7, GATS and GAGr are compared. The obtained results show that GATS performs significantly better than GAGr on the railway scheduling problems ; which indicates that the TS operator allows GATS to dig deeper in the search space and to find better solutions.

On the other hand, GATS is not able to reach the best-know solutions that were found by Lagrangian-based approaches. In fact, Lagrangian-based approaches use information found from the Lagrangian relaxation (reduced costs and column pricing) to significantly reduce the size of the problem solved. Reducing the size of the problem solved greatly reduces the computation-time needed to find an optimal or near-optimal solution.

6.4.4 Unicost problems

The detailed results of GATS on the unicost problems are presented in Table 6.8. We observe that GATS performs remarkably well on the USCP. In fact, GATS is able to find the best-known solutions of 19 of the 22 test problems and improves the best-known solutions for two of the test problems (CYC10 and STS135).

In Table 6.9, GATS is compared to three of the most popular metaheuristic approaches for the USCP. These metaheuristics are Meta-RaPS, a metaheuristic that is based on electromagnetism (EM) [73] and a local search algorithm (LS-USCP) [72]. The best solutions found by each algorithm and the average solution times are presented. All algorithms are executed on similar hardware, and the obtained solution times are similar. The hit frequencies are not included because none of the authors of the presented approaches have provided detailed results. The solution values presented for GATS, EM and USCP are the best of ten trials. Meta-RaPS does not provide sufficient information on the number of trials performed nor the hit frequencies for the unicost problems, which makes it difficult to draw conclusions

Table 6.6 Detailed results for the railway scheduling problems

		% deviation from BKS							
Problems	BKS	min	max	avg	SD	HF	T_{avg}(s)	Generations	Generations/s
RAIL507	174	2.3	3.45	3.1	0.7	0	65535.6	46107.6	6.4
RAIL516	182	1.1	2.2	1.7	0.57	0	45048.1	52439	7.28
RAIL582	211	3.32	4.27	4.08	0.7	0	51832.6	43955.4	6.1
RAIL2536	691	12.3	13.46	12.91	2.9	0	58978.3	1054.9	0.15
RAIL2586	948	9.92	10.97	10.46	2.49	0	44853.5	962.9	0.13
RAIL4284	1065	15.59	17.56	16.48	5.7	0	46619	628.5	0.09
RAIL4872	1534	14.02	15.91	15.29	9.58	0	41592.1	828.9	0.12

This table is similar in format to Table 6.2.

Table 6.7 Comparison of GATS and GAGr on the railway scheduling problems

		% deviation from BKS				T_{avg}(s)	
		GATS		GAGr		GATS	GAGr
Problems	BKS	min	avg	min	avg		
RAIL507	174	2.3	3.45	3.45	4.25	65535.6	59802.4
RAIL516	182	1.1	2.2	4.95	6.15	45048.1	65260.2
RAIL582	211	3.32	4.27	6.64	8.63	51832.6	64387.3
RAIL2536	691	12.3	13.46	18.67	18.99	58978.3	65679.2
RAIL2586	948	9.92	10.97	15.4	16.05	44853.5	60744.2
RAIL4284	1065	15.59	17.56	23.76	24.32	46619	70510.6
RAIL4872	1534	14.02	15.91	19.23	19.67	41592.1	65039

BKS is the best-known solution value before GATS and $T_{avg}(s)$ is the average solution-time in seconds.

based on the quality of their results. In comparison to LS-USCP, which is to our knowledge the best metaheuristic approach for the USCP, GATS finds equal solutions for 18 problems, better solutions for two problems (CYC10 and STS135) and worse solutions for two problems (CYC09 and CYC11). As a result, GATS is one of the best metaheuristic approaches for the USCP.

6.5 Summary

We presented a new hybrid metaheuristic that is based on the genetic algorithm and tabu-search (GATS) to solve the SCP. In addition to its effectiveness, an important characteristic of GATS is its simplicity. GATS is simple to understand and implement and does not involve sophisticated techniques for adjusting the tabu-list size. In fact, we use the simple mutation and crossover operators of the GA to adaptively adjust the tabu-list size. GATS is also simple to use because it does not require any significant adjustments of the parameters for specific problems.

We demonstrated the effectiveness of GATS on four classes of set covering problems. In our experiments, GATS outperformed all other metaheuristic approaches for the SCP on OR-Library problems, which are the most popular set covering problems for testing metaheuristic approaches. GATS also performed better than most metaheuristic approaches for the USCP (except for LS-USCP, which finds similar results with averagely shorter solution times). In addition, we tested GATS on real-world scheduling problems that have not been previously solved with metaheuristic-only approaches. GATS performed well on the airline and bus scheduling problems by finding the best know solution of thirteen of the sixteen test problems. On the other hand, GATS was not able to match the performance of Lagrangian-based approaches on the large-scale railway scheduling problems. We encourage future researcher to consider the airline, bus and especially the railway scheduling problems to test modern

Table 6.8 Detailed results for the unicost problems

Problems	BKS	% deviation from BKS min	max	avg	SD	HF	T_{avg}(s)	Generations	Generations/s
E1	5	0	0	0	0	1	0	2652710.2	368.4
E2	5	0	0	0	0	1	0	2650123.2	368.1
E3	5	0	0	0	0	1	0	2946560.2	409.2
E4	5	0	0	0	0	1	0	2741252	380.7
E5	5	0	0	0	0	1	0	2315623	321.6
CLR10	25	0	0	0	0	1	0	3932843.2	546.2
CLR11	23	0	0	0	0	1	0.4	3038169	422.0
CLR12	23	0	0	0	0	1	6.4	1504968.8	209.0
CLR13	23	0	0	0	0	1	309.8	996382	138.4
CYC06	60	0	0	0	0	1	0	4763774.8	661.6
CYC07	144	0	0	0	0	1	13.8	2787534.2	387.2
CYC08	342	0	2.92	0.91	3.84	0.5	676.3	906959	126.0
CYC09	774	2.84	5.94	4.86	9.11	0	930.4	338470	47.0
CYC10	1820	-0.11	1.76	1.36	10.02	0.1	2426.1	81174	11.3
CYC11	4088	2.05	2.69	2.4	7.42	0	4563	10711.2	1.5
STS9	5	0	0	0	0	1	0	607758667.6	84410.9
STS15	9	0	0	0	0	1	0	336336940	46713.5
STS27	18	0	0	0	0	1	0	179833191	24976.8
STS45	30	0	0	0	0	1	0	74288708.2	10317.9
STS81	61	0	0	0	0	1	0	25819803.8	3586.1
STS135	104	-0.96	-0.96	-0.96	0	1	7.89	12363867.8	1717.2
STS243	198	0	0	0	0	1	0	5282510.6	733.7

This table is similar in format to Table 6.2.

Table 6.9 Unicost problems

	Best solutions				Average solution-times			
Instance	GATS	EM	Meta-RaPS	LS-USCP	GATS Intel 2.67G	EM Intel 1.7GHZ	Meta-RaPS Intel 1.8GHZ	LS-USCP Intel 2.4GHZ
E1	5	5	5	5	0	0	0	1
E2	5	5	5	5	0	0	0	1
E3	5	5	5	5	0	0	0	1
E4	5	5	5	5	0	0	0.12	1
E5	5	5	5	5	0	0	0	1
CLR10	25	25	25	25	0	0.57	0	0
CLR11	23	23	23	23	0.4	15.53	3.03	0
CLR12	23	23	23	23	6.4	109.69	4.13	3.7
CLR13	23	23	23	23	309.8	3539.45	48.74	79
CYC06	60	60	60	60	0	0	0	0
CYC07	144	144	144	144	13.8	1.97	0	0
CYC08	342	344	344	342	676.3	303.4	38.91	11.1
CYC09	796	812	793	774	930.4	407.63	88.36	110.4
CYC10	1818	1915	1826	1820	2426.1	1892.06	80.56	488.9
CYC11	4172	4272	4140	4088	4563	12922.03	12656.75	1497.8
STS9	5	-	-	-	0	-	-	-
STS15	9	-	-	-	0	-	-	-
STS27	18	-	-	18	0	-	-	0
STS45	30	-	-	30	0	-	-	0.1
STS81	61	-	-	61	0	-	-	0
STS135	103	-	-	104	7.89	-	-	1.1
STS243	198	-	-	198	0	-	-	29.6

Meta-RaPS refers to [61], EM refers to [73] and LS-USCP refers to [72].

metaheuristic approaches for the SCP.

CHAPITRE 7

RECOUVREMENT PARTIEL AVEC BUDGET

Il arrive fréquemment des cas où le budget permis dans le problème de positionnement des trous de forages est limité. Ceci n'implique pas que le budget doit être dépensé au total, mais que le coût de la solution ne doit pas dépasser le budget permis. Pour tenir compte de ces cas, une contrainte de budget a été ajoutée au modèle PMSCP. Le modèle résultant est appelé BPMSCP (pour *budgeted profit maximization set covering problem*).

7.1 Le modèle BPMSCP

La formulation du BPMSCP est identique à celle du PMSCP, mais inclut en plus la contrainte de budget. Cette formulation est présentée ci-dessous.

Soit

— $A^{m\times n}$ une matrice binaire où $a_{ij} = 1$ si l'élément e_i est couvert par le sous-ensemble s_j, et $a_{ij} = 0$ autrement.

— $B^{l\times n}$ une matrice binaire où $b_{kj} = 1$ si le sous-ensemble s_j appartient au groupe G_k, et $b_{kj} = 0$ autrement.

— $X = \{x_1, x_2, ..., x_n\}$ où $x_j = 1$ si le sous-ensemble s_j (ayant un coût $c_j > 0$) est utilisé dans la solution, et $x_j = 0$ autrement.

— $Y = \{y_1, y_2, ..., y_m\}$ où $y_i = 1$ si l'élément e_i (ayant un gain $g_i > 0$) est couvert dans la solution, et $y_i = 0$ autrement.

— $GR = \{gr_1, gr_2, ..., gr_l\}$ où $gr_k = 1$ si au moins un des sous-ensembles du groupe G_k (ayant un coût $f_k > 0$) est utilisé dans la solution, et $gr_k = 0$ autrement.

— B le budget.

Maximiser

$$\sum_{i=1}^{m} g_i y_i - \sum_{j=1}^{n} c_j x_j - \sum_{k=1}^{l} f_k gr_k \tag{7.1}$$

Sujet à

$$y_i \leq \sum_{j=1}^{n} a_{ij} x_j, \qquad i = 1, ..., m \tag{7.2}$$

$$b_{kj} x_j \leq gr_k, \quad j = 1, ..., n; k = 1, ..., l \tag{7.3}$$

$$\sum_{j=1}^{n} c_j x_j + \sum_{k=1}^{l} f_k gr_k \leq B \tag{7.4}$$

$$x_j, y_i, gr_k \in \{0, 1\} \tag{7.5}$$

Il est important de faire la distinction entre le PMSCP avec budget et le *budgeted maximum coverage location problem (BMCLP)* discuté dans la revue de littérature. Le dernier implique la maximisation du gain total des blocs couverts tout en respectant une contrainte de budget. Un tel modèle peut mener à des solutions extrêmes où le coût de forage investi pour couvrir certains blocs est très élevé par rapport à la valeur qu'apportent ces blocs au forage de définition. Ce cas a été discuté à la section 2.2 où il a été montré que le modèle BMCLP n'est pas adéquat pour le problème PTF. Comme dans le PMSCP, le PMSCP avec budget (BPMSCP) permet d'éviter les solutions extrêmes. L'objectif n'est pas de maximiser la somme des gains des blocs couverts, mais de maximiser le profit représenté par la somme des gains des blocs couverts moins le coût des trous utilisés et les frais de positionnement de la foreuse.

Pour résoudre le BPMSCP, nous avons adapté l'algorithme ITS pour prendre en compte la contrainte de budget.

7.2 ITS avec budget

ITS avec budget ou BITS (pour *budgeted-iterated-tabu-search*) est une adaptation directe de ITS où les opérateurs de perturbation et de recherche locale ont été adaptés pour prendre en compte la contrainte de budget.

Dans le nouvel opérateur de recherche taboue avec budget (BTS), les mouvements qui résultent d'une solution dont le coût est supérieur au budget sont interdits. Le pseudo-code de BTS est présenté à l'algorithme 8 et 9 où δ_i est la variation de la fonction objectif qui résulte de l'ajout ou du retrait d'un sous-ensemble s_i, c_i est le coût d'un sous-ensemble s_i et B est le budget. La seule différence entre BTS et TS est l'ajout de la ligne 7 au pseudo-code de l'algorithme 9.

L'utilisation de BTS au lieu de TS dans ITS n'est pas suffisante pour assurer que la contrainte de budget est respectée. En fait, le premier opérateur de perturbation de ITS peut engendrer des solutions dont le coût est supérieur au budget. Cet opérateur retire et ajoute des

Algorithm 8 BTS(X)

```
1: clear-tabu-lists() ;
2: best ← X ;
3: stagnationCounter ← 0 ;
4: loop
5:    m ← find-next-move(X, objective-value(best)) ;
6:    if (X_m = 0) then
7:       add(X, m) ;
8:    else
9:       remove(X, m)
10:   end if
11:   mark-tabu(m) ;
12:   if (score(X) > score(best)) then
13:      best ← X ;
14:      stagnationCounter ← 0 ;
15:   else if (stagnationCounter > nStag) then
16:      return ;
17:   end if
18:   stagnationCounter ← stagnationCounter + 1 ;
19: end loop
```

Algorithm 9 find-next-move(X, bestScore)

```
1: score = objective-value(X)
2: δ_max ← −∞
3: m ← 0
4: for i = 1 : numberOfSets(X) do
5:    if ( (isTabu(i) = False) or (score + δ_i > bestScore) ) then
6:       if (δ_i > δ_max) then
7:          if ( (X_i = 1) or (cost(X) + c_i ≤ B) ) then
8:             δ_max ← δ_i
9:             m ← i
10:         end if
11:      end if
12:   end if
13: end for
14: return m
```

sous-ensembles à la solution d'une façon aléatoire. Par conséquent, la solution résultante peut avoir un coût supérieur au budget. Pour éliminer cette possibilité, l'opérateur de perturbation a été remplacé par un opérateur de perturbation unidirectionnel. Cet opérateur itère sur tous les sous-ensembles de la solution et retire chaque sous-ensemble d'une façon probabiliste, tel que montré à l'algorithme 10. L'utilisation de cet opérateur de perturbation et de l'opérateur BTS est suffisante pour assurer que la contrainte de budget est respectée dans ITS.

Algorithm 10 BITS-Perturb(X)

for ($i = 1$: numOfSets(X)) **do**
  if (flip-coin(p) = true) **then** {p *is the perturbation strength*}
    $X \leftarrow$ remove-set(X, i)
  end if
end for

Notons qu'il est aussi possible d'implémenter la contrainte de budget en utilisant une approche de pénalités où les solutions non-réalisables sont pénalisées dans la fonction objectif. Nous avons choisi l'approche présentée dans ce chapitre parce qu'elle est simple et efficace. En fait, des tests préliminaires ont montré que cette approche est très efficace. De plus, l'algorithme BITS a été testé sur des cas tests réels présentés au chapitre 9.

CHAPITRE 8

PRÉ-CONDITIONNEMENT ET PARAMÉTRISATION DU PROBLÈME PTF

8.1 Assignation des gains aux blocs

Tel que mentionné précédemment, pour résoudre le problème PTF comme un PMSCP ou un BPMSCP, un gain doit être attribué à chaque bloc de minéralisation. D'ailleurs, l'objectif à optimiser est défini par les valeurs de ces gains. Dans la revue de littérature (section 2), nous avons vu que Pan [76] propose une approche géostatistique complète qui permet de classifier les blocs de minéralisation dans le modèle de blocs en leur assignant des valeurs qui reflètent leurs priorités par rapport au forage de définition. Ces valeurs dépendent à la fois de la teneur en minerai des blocs et de l'incertitude sur cette teneur.

Nous proposons de réutiliser l'approche de Pan et d'assigner un gain à chaque bloc selon la classe à laquelle ce bloc appartient. Les blocs appartenant aux classes cibles importantes pour le forage de définition auront des gains élevés et ceux appartenant aux classes moins importantes auront des gains moins élevés. Ceci permet de compléter le travail débuté par Pan en ajoutant la capacité d'optimiser la position des trous en 3D offerte par notre approche.

Une approche plus simple permettant d'assigner les gains aux blocs est d'attribuer à chaque bloc un gain égal au coût qu'on est prêt à investir pour couvrir ce bloc. L'utilisation de tels gains fait en sorte qu'un trou est considéré comme candidat dans la solution si et seulement si la somme des gains des blocs couverts par ce trou est supérieure au coût du trou (le trou est profitable).

Il est aussi possible et parfois souhaitable de laisser au géologue la liberté d'assigner ou modifier les gains pour contrôler l'objectif à optimiser. Ces gains offrent aux géologues une grande flexibilité en leur permettant de définir d'une façon très précise l'objectif à optimiser. Pour qu'une telle approche soit efficace, un outil graphique permettant aux géologues d'assigner des gains aux blocs d'une façon facile et rapide est nécessaire. Une première version d'un tel outil a été développé par la compagnie *Objectivity* qui est partenaire dans ce projet. L'outil permet aussi au géologue de manipuler l'univers des trous en supprimant des trous (par exemple les trous qui semblent irréalisables), modifier les coûts assignés aux trous, etc.

Plusieurs scénarios d'utilisation du PMSCP dans le cadre du problème PTF sont discutés dans la section 8.2 qui suit. L'assignation des gains aux blocs permettant l'atteinte de l'objectif de chaque scénario est discutée.

8.2 Scénarios d'utilisation

Les modèles d'optimisation développés dans cette thèse ont été conçus pour être flexibles et adaptables à plusieurs situations. En ajustant les gains attribués aux blocs, les coûts fixes attribués aux déplacements de la foreuse, les coûts des trous, le rayon de couverture des trous et le budget permis, les modèles peuvent être utilisés pour atteindre un grande variété d'objectifs possibles. Ceci permet l'utilisation d'un même modèle et algorithme d'optimisation pour résoudre plusieurs problèmes.

Dans cette section, nous présentons plusieurs scénarios d'utilisation des modèles d'optimisation pour atteindre différents objectifs. Ces scénarios illustrent des exemples concrets d'utilisation de nos modèles pour répondre à différents besoins qui arrivent dans la planification des trous de forage. Ces scénarios peuvent aussi être combinés pour créer des nouveaux scénarios plus complexes.

Scénario 1 : Couvrir tous les blocs.

L'objectif est de couvrir tous les blocs à un coût minimal. Le coût de positionnement de la foreuse n'est pas important et le budget est illimité.

Ce problème est clairement un problème de recouvrement d'ensembles classique (SCP) mais peut aussi être modélisé sous forme d'un PMSCP ou un BPMSCP.

Pour modéliser le problème sous forme d'un PMSCP, il suffit d'utiliser des gains suffisamment élevés et des coûts de positionnement de la foreuse nuls. Soit c_i^{min} le coût du sous-ensemble le moins cher qui couvre un élément e_i donné ; pour s'assurer que tous les blocs seront couverts, le gain g_i attribué à chaque sous-ensemble e_i doit être supérieur au c_i^{min} associé à cet élément. Une façon plus simple d'attribuer les gains aux éléments dans ce cas est de calculer le coût du trou le plus cher dans l'univers des trous (c^{max}) et d'assigner un gain supérieur à c^{max} à tous les blocs.

La même approche peut être utilisée pour modéliser ce problème sous forme d'un BPMSCP, en utilisant un budget infini.

Même si la solution optimale est la même pour les trois modélisations, dans ce cas, il est préférable de modéliser le problème sous forme d'un SCP puisque ce dernier est un cas particulier des deux autres. Résoudre le cas particulier est généralement plus facile parce que l'espace de recherche est réduit.

Scénario 2 : Couvrir tous les blocs et minimiser le repositionnement de la foreuse.

Dans certains cas, le nombre de positions possibles de la foreuse est très élevé et nous voulons minimiser le nombre de positions utilisées pour éviter des situations où la foreuse doit être repositionnée très souvent. L'objectif est de couvrir tous les blocs à un coût minimal tout en minimisant le repositionnement de la foreuse. Le budget est illimité.

Même si tous les blocs doivent être couverts, le SCP n'est pas adéquat pour modéliser cette situation parce que le SCP ne prend pas en compte les coûts de repositionnement de la foreuse. Le PMSCP est un meilleur candidat que le BPMSCP dans ce cas parce que le budget est illimité.

Pour s'assurer que tous les blocs seront couverts, les gains attribués aux éléments (blocs) doivent être suffisamment élevés. Soit e_i un élément donné. Pour chaque sous-ensemble s_j qui couvre l'élément e_i, soit $C_j = c_j + f_j$ le coût du sous-ensemble s_j additionné au coût du groupe auquel s_j appartient. Le gain attribué à chaque élément e_i doit être supérieur au plus petit C_j de tous les sous-ensembles qui couvrent cet élément.

Scénario 3 : Couvrir un sous-ensemble de blocs

Seulement un sous-ensemble des blocs doivent être couverts. Le budget est illimité et le coût de repositionnement de la foreuse n'est pas important.

Ce scénario peut être modélisé sous forme d'un SCP ou d'un PMSCP. Pour utiliser le SCP, il suffit d'éliminer tous les blocs qui ne doivent pas être couverts du modèle de blocs et de garder seulement les blocs à couvrir. Pour utiliser le PMSCP, il suffit d'assigner un gain nul aux blocs qui ne doivent pas être couverts et un gain suffisamment élevé aux blocs à couvrir tel que discuté dans le scénario 1.

Scénario 4 : La zone interdite

Pour certaines raisons géologiques, certains blocs ne doivent pas être couverts par des trous.

Pour s'assurer que ces blocs ne soient pas couverts par des trous, il suffit d'utiliser le PMSCP et d'attribuer des gains négatifs très bas aux blocs ($g_i = -\infty$ pour tout bloc e_i appartenant à la zone interdite). Avec des tels gains, couvrir un des blocs appartenant à la zone interdite causera une grande chute du score de la solution et par conséquent, ces blocs seront évités.

Une autre façon de s'assurer que les blocs de la zone interdite ne soient pas couverts par des trous est d'éliminer tous les trous qui couvrent ces blocs de l'univers des trous. Cette deuxième méthode est plus efficace parce qu'elle permet de réduire la taille du problème.

Scénario 5 : Forage d'exploration

Avec un budget limité, le but est de répartir les trous de forage le plus uniformément possible dans le modèle de blocs. Le budget ne permet de couvrir qu'une petite partie des blocs.

Une approche itérative peut être utilisée pour résoudre ce scénario. Puisque le budget est limité, le BPMSCP est le modèle à utiliser dans ce cas. Premièrement, comme dans le scénario 1, des gains élevés sont utilisés pour mener l'algorithme d'optimisation à couvrir le

plus de blocs possible. Puisque le but est de répartir les trous le plus uniformément possible dans le modèle de bloc, le rayon de couverture n'est pas connu à l'avance et est déterminé d'une façon itérative. Pour commencer, un rayon est choisi et une solution initiale est trouvée. Trois résultats sont possibles :

1. La solution ne couvre pas tous les blocs parce que le budget est limité. Dans ce cas, il suffit d'augmenter le rayon de couverture des trous et résoudre le problème à nouveau. En augmentant le rayon de couverture, la quantité de forage nécessaire pour couvrir tous les blocs est moins élevée et par conséquent, le budget requis pour couvrir tous les blocs est moins élevé.
2. La solution couvre tous les blocs mais le coût de la solution est significativement inférieur au budget. Ceci indique que la répartition des trous peut être améliorée. Dans ce cas, il suffit de diminuer le rayon de couverture et de résoudre à nouveau.
3. La solution couvre tous les blocs (ou presque) et tout le budget (ou presque) a été dépensé. Ceci indique que le rayon choisi est adéquat et que les trous ont bien été répartis (en supposant que l'algorithme d'optimisation a convergé vers une solution optimale ou quasi-optimale).

   Généralement, il suffit de lancer des courtes exécutions de l'algorithme d'optimisation jusqu'à ce que un rayon adéquat soit trouvé. Par la suite, en utilisant le rayon trouvé, l'algorithme d'optimisation peut être lancé plus longtemps pour améliorer la qualité de la solution.

Scénario 6 : Forage de définition avec budget

Avec un budget limité, le but est de couvrir les blocs les plus pertinents pour la planification de l'exploitation minière. Les coûts de déplacements de la foreuse doivent être pris en compte.

Ce scénario représente le cas le plus général du forage de définition traité dans cette thèse, le BPMSCP. Le gain attribué à chaque bloc est proportionnel au coût qu'on est prêt à investir pour couvrir ce bloc, le coût attribué à un sous-ensemble est égal au coût du forage du trou correspondant et le coût fixe attribué à un groupe est égal au coût de positionnement de la foreuse à la position correspondante. Tel que mentionné précédemment, l'approche de Pan [76] peut aussi être utilisée pour l'assignation des gains aux blocs dans ce cas.

CHAPITRE 9

CAS TESTS MINIERS

Dans ce chapitre, nous présentons quatre cas tests réels provenant de l'industrie minière. Nous comparons les solutions trouvées par nos algorithmes d'optimisation aux solutions trouvées par les géologues en industrie. Pour des raisons de confidentialité, les noms des compagnies minières qui ont aidé à la réalisation de ces cas tests sont parfois omis.

9.1 Cas test 1

Dans ce premier cas test, le modèle de blocs est formé de 514560 blocs où chaque bloc a une dimension de $5 \times 5 \times 5$ mètres. Trois positions de la foreuse (points de collet) sont permises, comme montré à la figure 9.1. Le rayon de couverture des trous est de 100 mètres et le but est de couvrir tous les blocs.

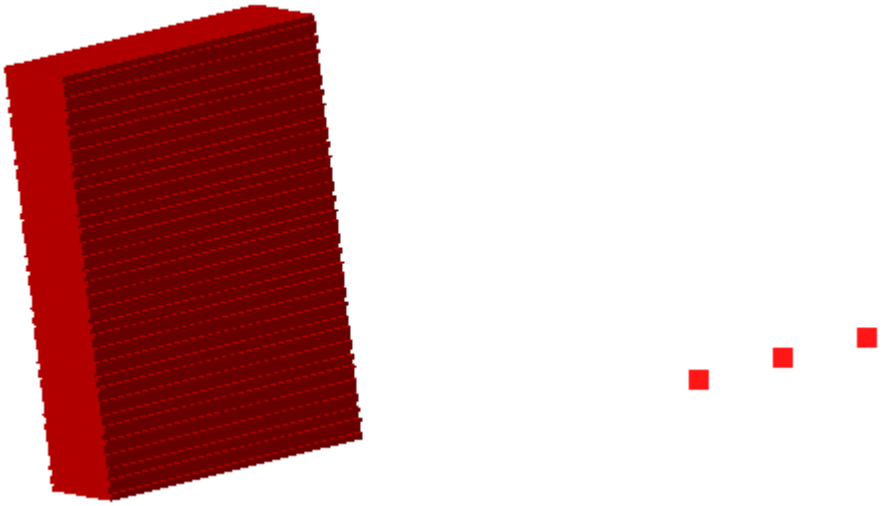

Figure 9.1 Cas test 1 : Modèle de blocs et positions permises de la foreuse

Solution des géologues

Suite à une discussion avec les géologues qui travaillent sur ce cas test, ces derniers ont suggéré de tracer une grille régulière espacée de 100×100 mètres sur la surface avant du modèle de blocs et de forer les trous reliant les points de collet et les points de la grille.

Pour chaque point de collet, les points de la grille considérés sont les points qui se trouvent à proximité de ce point (les points de la grille qui sont plus proches de ce point de collet que des autres deux points de collet). La longueur d'un trou est délimitée par les intersections du trou avec le modèle de blocs (un trou doit être suffisamment long pour couvrir au moins un bloc, mais suffisamment court pour ne pas dépasser le modèle de blocs et forer au-delà de la zone d'intérêt). La solution suggérée par les géologues est montrée à la figure 9.2. La longueur totale de forage est de 71375 mètres et couvre 99.12% des blocs.

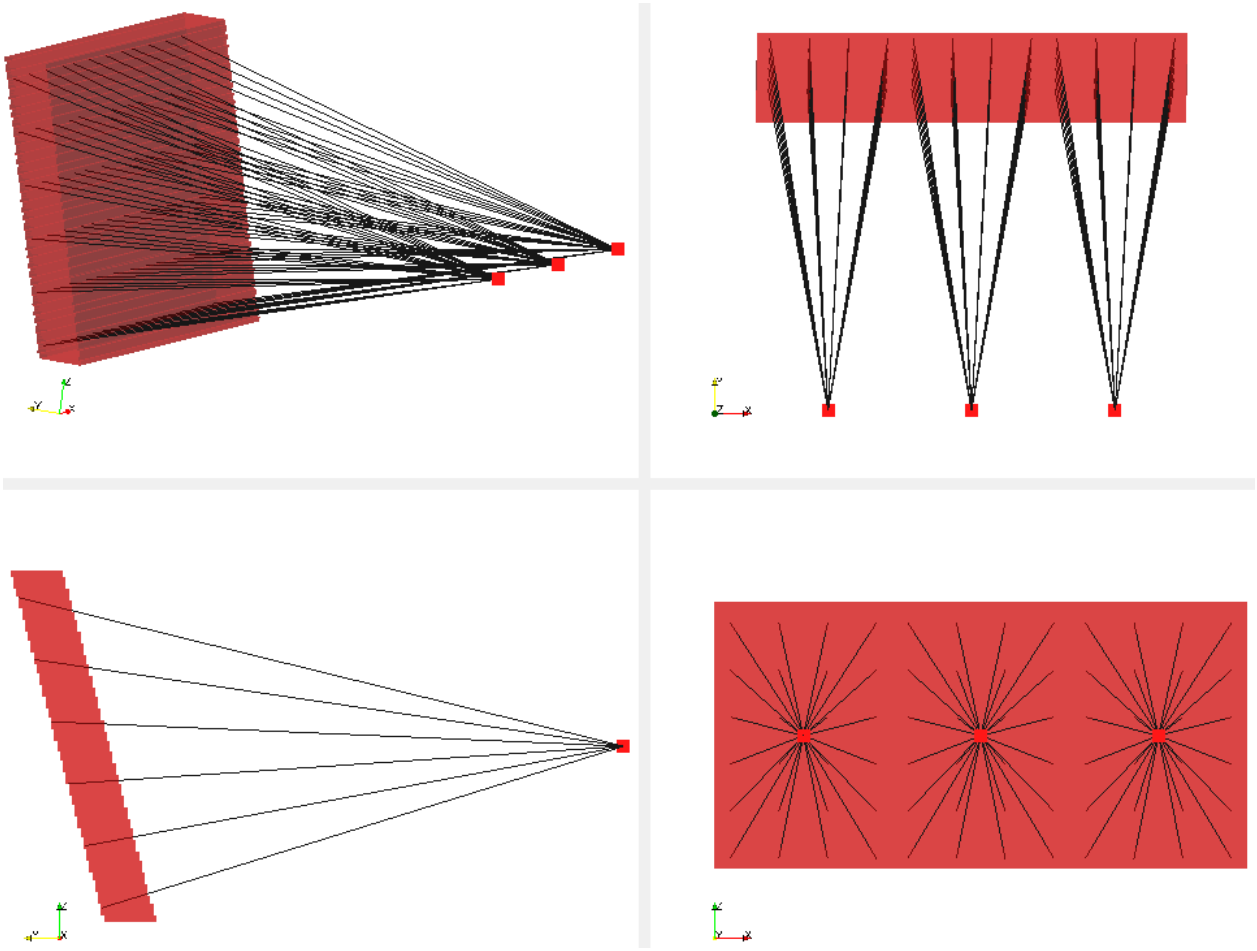

Figure 9.2 Cas test 1 : Solution suggérée par les géologues

Solution de l'optimiseur :

Ce problème a été modélisé comme un SCP et résolu à l'aide de l'algorithme GATS présenté au Chapitre 6.

Pour générer l'univers des trous, nous avons utilisé une précision en longueur de L_{step}=10 mètres (voir Algorithme 1), une précision en dip de Dip_{step}=2 degrés et une précision en

azimut de Az_{step}=10 degrés. La précision en azimut utilisée est plus élevée que la précision en dip parce que selon les géologues, la rotation de la foreuse en azimut est beaucoup plus coûteuse que la rotation en dip. L'univers des trous générés contient 78860 trous candidats et est montré à la figure 9.3.

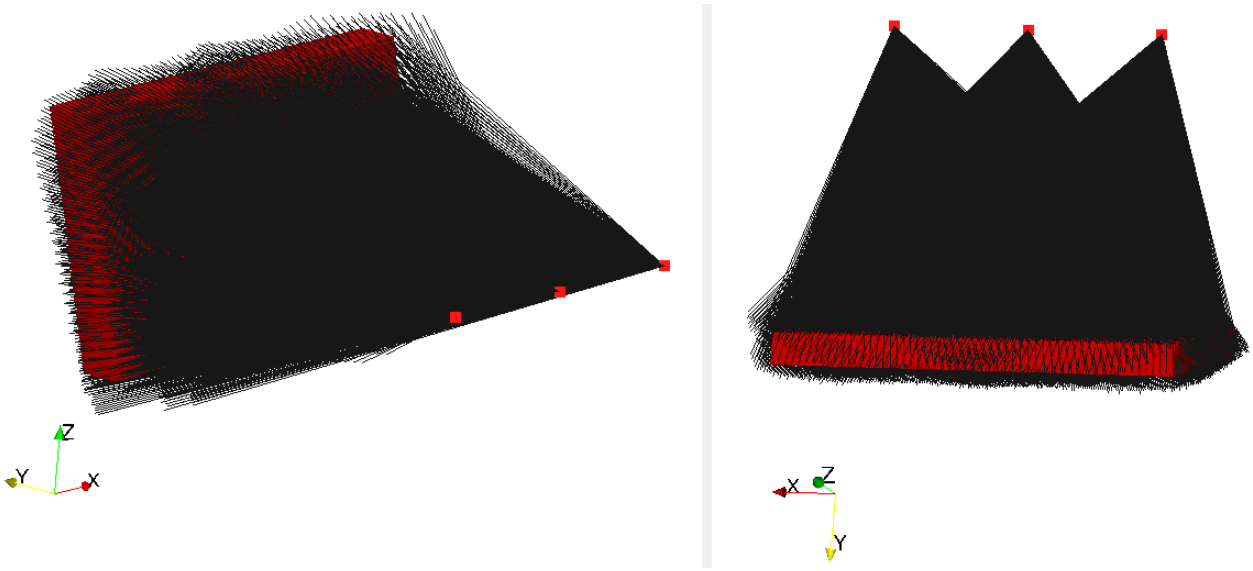

Figure 9.3 Cas test 1 : Univers des trous candidats

À cause de la grande taille du problème, l'algorithme GATS a été exécuté pour 48 heures. Ce temps est considéré raisonnable dans l'industrie des trous de forage parce que la planification des trous est une activité peu fréquente et est généralement effectuée une seule fois. Une mise à jour de la planification des trous peut arriver de temps en temps, mais le temps qui sépare deux planifications d'une même campagne de forage est généralement en mois ou années. La solution trouvée par GATS est montrée à la figure 9.4. Cette solution couvre tous les blocs en utilisant une longueur totale de forage de 46090 mètres, soit 35.4% moins de forage que la solution suggérée par les géologues ; ce qui représente une haute réduction du coût de forage.

9.2 Cas test 2

Dans le deuxième cas test, un modèle de blocs de 492109 blocs est utilisé où chaque bloc a une dimension de $5 \times 5 \times 5$ mètres. Douze positions possibles de la foreuse sont disponibles et les coûts de repositionnement de la foreuse ne sont pas considérés. Le rayon de couverture des trous est de 100 mètres et le but est de couvrir le plus de blocs possible tout en respectant un budget limité à 23590 mètres. Ce problème est un problème réel où les trous ont déjà été

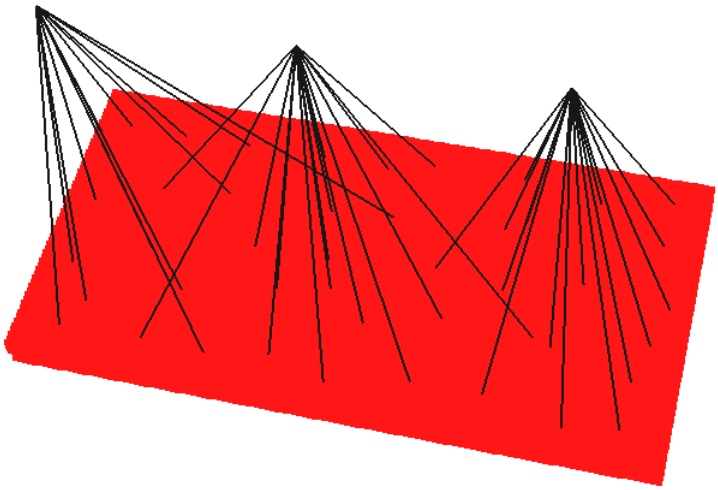

Figure 9.4 Cas test 1 : Solution trouvée par l'optimiseur

choisis et forés. Nous résolvons le même problème à l'aide de notre approche d'optimisation et comparons notre solution à la solution existante.

Nous modélisons le problème sous forme d'un BPMSCP. Puisque le but est de couvrir le plus de blocs possible, les gains attribués aux blocs sont égaux et sont choisis suffisamment élevés, tel que discuté dans le scénario 1 de la section 8.2. Le coût attribué aux trous est égal à leur longueur parce que le budget est en mètre, et le coût des groupes est égal à zéro parce que les frais de repositionnement de la foreuse ne sont pas pris en compte. L'algorithme utilisé est BGATS.

La solution existante et la solution trouvée par BGATS sont présentées à la figure 9.5. Tel que montré dans la figure, la solution existante couvre 47.3% des blocs tandis que la solution trouvée par notre approche couvre 74.7% des blocs ; ce qui représente une amélioration importante de la couverture (27.4% de blocs additionnels sont couverts).

9.3 Cas test 3

Le troisième cas test est similaire au premier où le but est de couvrir tous les blocs à un coût minimal et le budget est illimité. Le nombre de blocs est 21713. Comme dans le premier cas test, le problème est modélisé sous forme d'un SCP et résolu à l'aide de GATS.

De même que le cas test précédent, ce cas test provient d'un problème réel où les trous ont déjà été forés. Pour évaluer notre approche, nous utilisons les mêmes blocs couverts par la solution existante et tentons de trouver une solution qui couvre ces mêmes blocs à un coût moins élevé.

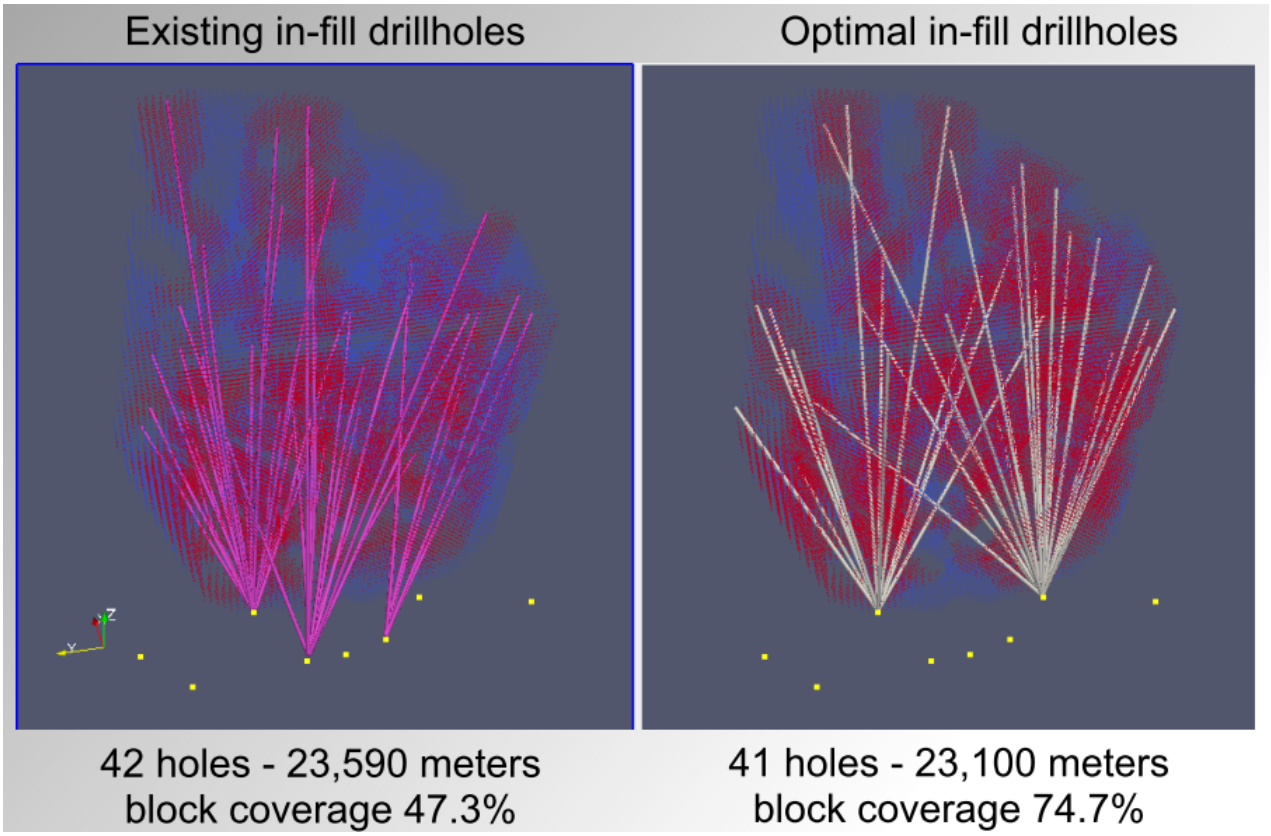

Figure 9.5 Cas test 2 : Comparaison des solutions

La solution existante et la solution trouvée par GATS sont présentées à la figure 9.6. La solution existante utilise 31343.9 mètres de forage tandis que notre solution n'utilise que 23520.05 mètres de forage ; ce qui représente une réduction de 25% des coûts de forage.

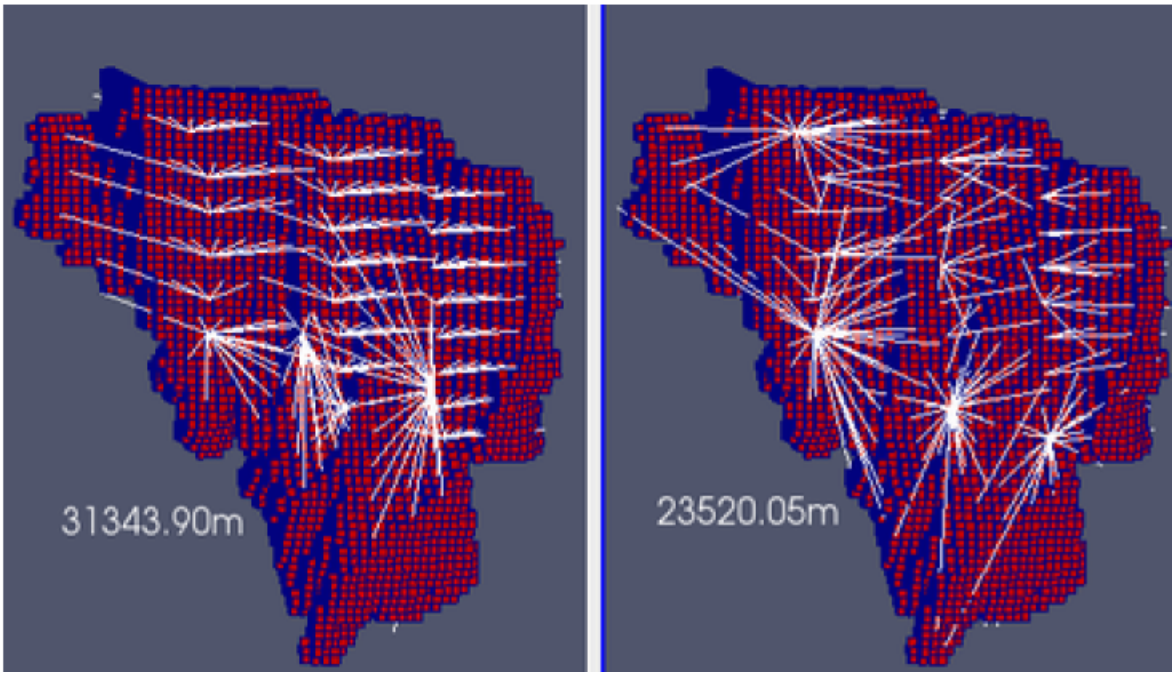

Figure 9.6 Cas test 3 : Comparaison des solutions

9.4 Cas test 4

Ce cas test a été réalisé à Royal Nickel Corporation (RNC) en Abitibi Mining Camp, 25 km au nord de Amos, Québec. L'étude a été menée par RNC et Objectivity (sans ma participation) en utilisant les algorithmes développés dans cette thèse.

Les géologues et ingénieurs de RNC ont réalisé un plan initial qui spécifie les objectifs et contraintes à respecter. Les contraintes incluent : un minimum d'espacement entre les trous, un nombre limité de positions où la foreuse peut être placée, des contraintes sur les angles d'inclinaison des trous et certaines zones interdites.

Pour assurer une évaluation non biaisée des solutions, la compagnie SRK Consulting (Canada) Inc. a été engagée pour évaluer les solutions.

Les critères d'évaluation sont le coût total de la solution, la longueur totale des trous, le nombre de positions de la foreuse utilisées et le rapport de couverture (le rapport du volume total couvert sur la longueur totale des trous en m^3/m).

Pour évaluer la performance de l'optimiseur, la solution proposée par ce dernier a été comparée à une solution développée par les experts de RNC.

Les résultats sont présentés dans le tableau 9.1 et à la figure 9.7.

Tableau 9.1 Cas test 4 : Royal Nickel Corporation

Solution	Trous	Points de collet	Longueur totale	Rapport de couverture (m^3/m)	Coût
Manuelle	64	64	9673	2975	-
Optimiseur	54	42	6421	4067	-455,280$

Les résultats montrent que la solution optimisée améliore le rapport de couverture de 36% tout en utilisant 33% moins de forage et 34% moins de points de collet. La validité de la solution optimisée a été vérifiée par les experts de RNC et SRK qui ont confirmé que la solution respecte toutes les contraintes.

Cette étude réalisée en industrie a montré que l'approche d'optimisation permet de réaliser des économies majeures par rapport à l'approche manuelle.

Un avantage important de notre approche souligné par l'étude RNC est la possibilité d'évaluer plusieurs solutions de forage rapidement en variant les contraintes et les objectifs (gains). Par exemple, pour évaluer le bénéfice d'augmenter le budget de forage pour un cas test donné, il suffit de relancer l'algorithme avec un budget plus élevé et comparer la solution trouvée à la solution actuelle (trouvée en utilisant un budget moins élevé). Par conséquent, notre approche est vue comme un outil d'aide à la prise de décision très utile pour évaluer différents plans d'investissement dans l'exploration minière.

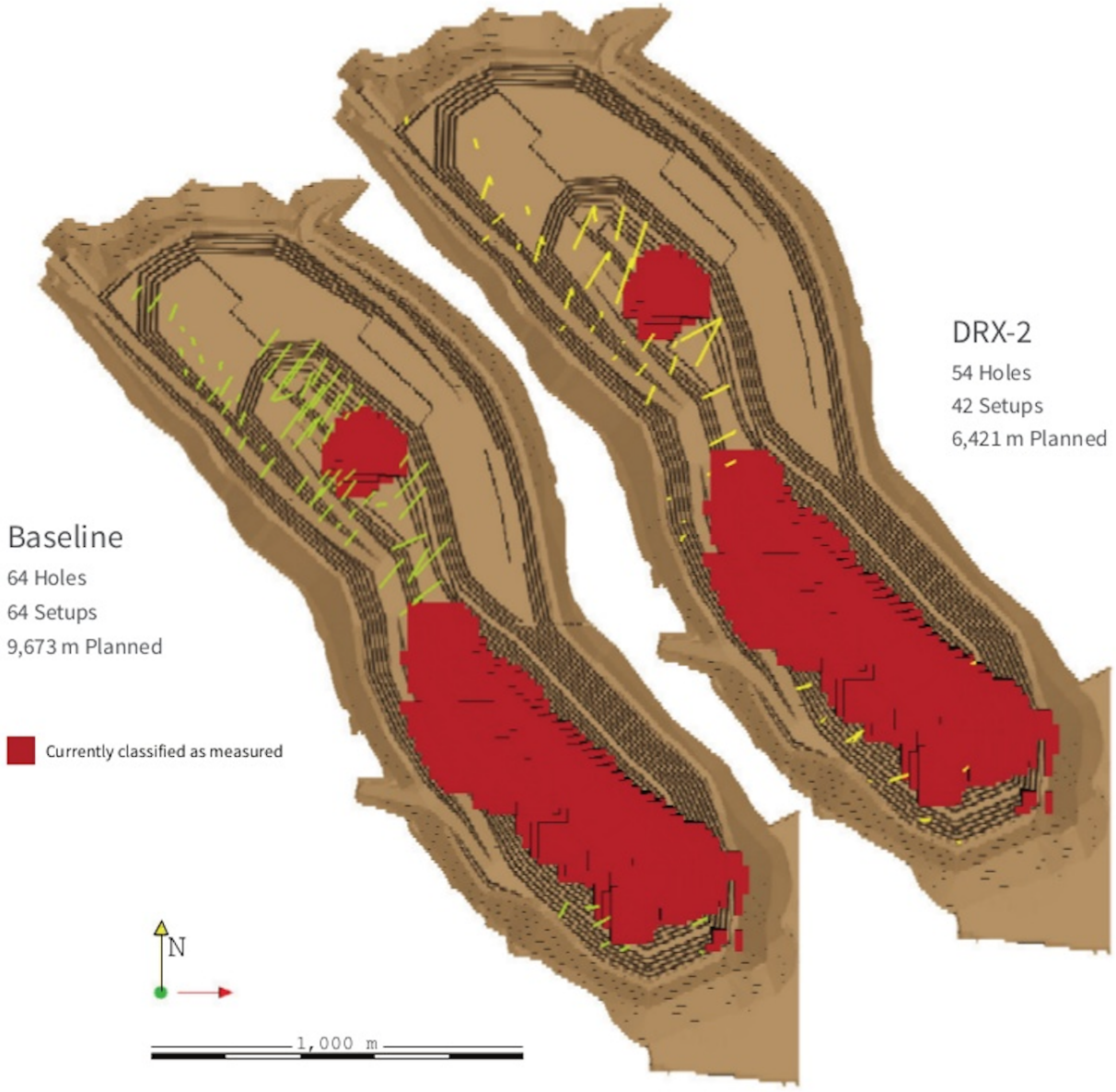

Figure 9.7 Cas test 4 : Royal Nickel Corporation

CHAPITRE 10

DISCUSSION GÉNÉRALE

Dans cette thèse, nous avons proposé une nouvelle approche pour résoudre le problème de positionnement des trous de forage dans les mines. La nouvelle approche utilise des modèles d'optimisation basés sur le problème de recouvrement d'ensembles. Pour résoudre les modèles proposés, nous avons développé des algorithmes métaheuristiques hybrides. La raison pour laquelle nous avons choisi de développer des algorithmes métaheuristiques au lieu d'utiliser un solveur de programmation mathématique comme CPLEX est la grande taille des problèmes. Tel que discuté au chapitre 4, sans modifications majeures, CPLEX ne permet pas de résoudre les problèmes de grande taille dans un temps raisonnable.

Dans la revue de littérature, nous avons identifié plusieurs limitations majeures des méthodes existantes utilisées pour résoudre le problème PTF. Parmi ces limitations :

1. Le problème est fréquemment traité en 2D au lieu de 3D.
2. Les algorithmes d'optimisation proposés sont peu efficaces et ne trouvent généralement pas la solution optimale du problème posé.
3. Les coûts de positionnement de la foreuse ne sont pas pris en compte.
4. La contrainte de budget n'est pas prise en compte.
5. Dans le cas 3D, les hypothèses suggérées ne sont pas tout à fait réalistes et compromettent l'optimalité des solutions. Parmi les hypothèses rencontrées :
   - Le nombre de trous est connu à l'avance.
   - La longueur des trous est supposée constante.
   - Les trous sont supposés parallèles.
   - L'azimut des trous est connu à l'avance.

En utilisant une approche basée sur le recouvrement d'ensembles, la géométrie des trous est définie à l'avance dans l'étape de génération de l'univers des sous-ensembles. L'approche de génération de l'univers des trous discutée à la section 3.1 permet de générer tous les trous possibles dans l'espace 3D selon la précision de la foreuse. Par la suite, l'algorithme d'optimisation est utilisé pour trouver une solution optimale ou quasi optimale. Par rapport aux limitations des approches existantes discutées ci-dessus, notre approche offre les avantages suivants :

1. Le problème est traité en 3D grâce au fait que l'ensemble de tous les trous possibles est généré en 3D. Par conséquent, toutes les directions des trous (selon la précision de la foreuse) sont prises en considération. De plus, la longueur des trous est optimisée parce que le générateur des trous crée des trous de différentes longueurs pour une direction donnée.
2. Le nombre de trous n'est pas connu à l'avance et est choisi par l'algorithme d'optimisation.
3. Les coûts de repositionnement de la foreuse sont pris en compte dans nos modèles PMSCP et BPMSCP.
4. La contrainte de budget est prise en compte dans notre modèle BPMSCP.
5. Nous avons montré dans les chapitres 4 et 6 que les algorithmes d'optimisation proposés sont très efficaces.

À la section 8.2, nous avons discuté plusieurs scénarios d'utilisation de notre approche permettant de répondre à plusieurs besoins qui peuvent intervenir durant la planification des trous de forage. Nous avons montré que notre approche peut être aisément adaptée à plusieurs situations sans modifier le modèle ou l'algorithme d'optimisation. L'adaptation est réalisée en ajustant les gains assignés aux blocs et les trous inclus dans l'univers des trous, qui représentent des paramètres du problème. Notons que les composantes de génération de l'univers des trous et de l'assignation des gains aux blocs sont indépendantes l'une de l'autre et du modèle d'optimisation et peuvent être abordées d'une façon indépendante. Bien que ces composantes aient été abordées dans cette thèse, elles peuvent être traitées d'une façon plus avancée dans une thèse en géostatistique. Par exemple, le générateur des trous pourrait être amélioré en y ajoutant la capacité de prévoir la trajectoire des trous et de générer des trous courbes. Par contre, il n'est actuellement pas possible de prévoir la trajectoire des trous de forage d'une façon précise.

En plus des tests réalisés sur les algorithmes aux chapitres 4 et 6, quatre cas tests réels ont été utilisés pour comparer notre approche aux approches qui sont présentement utilisées dans l'industrie minière (voir chapitre 9). Par rapport à ces approches, notre approche a permis de :

— réduire les coûts de forage de 35.4% dans le premier cas test,

— améliorer la couverture des blocs de 27.4% dans le deuxième cas test,

— réduire les coûts de forage de 25% dans le troisième cas test et,

— améliorer la couverture des blocs de 36% en utilisant 33% moins de forage et 34% moins de points de collet dans le quatrième cas test.

Les résultats obtenus montrent que les modèles proposés sont très convenables pour modéliser le problème PTF et que les algorithmes d'optimisation développés sont très efficaces pour résoudre ces modèles. Par conséquent, l'approche proposée dans cette thèse est très efficace pour résoudre le problème PTF dans l'exploration minière.

Outre les contributions apportées à l'exploration minière, nous avons réalisé des contributions importantes au SCP et au PSCP. Au chapitre 4, nous avons proposé une nouvelle variante du recouvrement partiel et un algorithme métaheuristique très efficace pour résoudre cette variante. Nous avons aussi montré que l'utilisation d'un deuxième opérateur de perturbation et d'un opérateur de recherche taboue dans la recherche locale itérée permet d'améliorer significativement la performance de cette métaheuristique. Au chapitre 5, nous avons proposé une nouvelle formulation du SCP. Cette formulation facilite considérablement la résolution du SCP avec des algorithmes métaheuristiques en éliminant les difficultés liées à la faisabilité des solutions et à la redondance des sous-ensembles. Finalement, au chapitre 6, nous avons développé un algorithme métaheuristique très efficace pour résoudre le SCP dont les performances sont meilleures que la majorité des algorithmes métaheuristiques existants. Cet algorithme a permis d'améliorer les meilleures solutions connues pour plusieurs problèmes populaires dans la littérature.

Bien que l'hypothèse de couverture binaire que nous avons posée à la Section 1.6 a été justifiée, il est possible de mieux exploiter l'aspect géométrique du problème PTF. En fait, dans nos modèles, nous avons considéré qu'un bloc est couvert par un trou si et seulement si ce bloc se trouve à une distance $r \leq R_{max}$ de ce trou (où R_{max} est un paramètre). En pratique, si un bloc est entouré par deux (ou plus) trous qui se trouvent à une distance légèrement supérieure à R_{max} de ce bloc, ce dernier peut-être considéré couvert. De plus, un bloc est mieux couvert par des trous qui se trouvent proche de côtés opposés (géométriquement) de ce bloc que par des trous qui se trouvent du même côté de ce bloc. Cet aspect géométrique du problème PTF sera étudié plus en détails dans nos travaux futurs.

CHAPITRE 11

CONCLUSIONS ET RECOMMANDATIONS

Dans cette thèse, nous avons développé une nouvelle approche pour résoudre le problème de positionnement des trous de forage dans les mines. La nouvelle approche est basée sur les problèmes de recouvrement et recouvrement partiel d'ensembles. Nous avons montré que notre approche permet d'éliminer la majorité des limitations des méthodes existantes et de trouver des solutions de meilleures qualités (une réduction des coûts de 25 à 35.4% a été obtenue dans nos expériences). Nous avons aussi montré que notre approche est flexible et adaptable à une variété de situations qui peuvent avoir lieu durant l'exploration minérale.

Par conséquent, en réponse à la question de recherche, nous pouvons conclure qu'il est possible et bénéfique de résoudre le problème de positionnement des trous de forage sous forme d'une variante du problème de recouvrement d'ensembles.

En plus des contributions apportées à l'exploration minière, nous avons réalisé des contributions importantes au SCP (et ses variantes). Puisque le SCP est un problème très populaire qui a été utilisé pour modéliser un grand nombre d'applications industrielles, ces contributions sont pertinentes.

Nos futurs travaux porteront sur la parallélisation des algorithmes développés pour réduire leur temps d'exécution et augmenter leur efficacité. La parallélisation de métaheuristiques est un domaine relativement nouveau et prometteur.

Une autre voie de recherche que nous considérons est l'utilisation de la Relaxation Lagrangienne dans nos algorithmes métaheuristiques. L'utilisation de la Relaxation Lagrangienne permet de réduire la taille de l'espace de recherche et par conséquent, accélère la convergence vers la solution optimale.

RÉFÉRENCES

[1] G. Abdel Azim and M. Ben Othman. Hybrid iterated local search algorithm for solving multiple sequences alignment problem. *Far East Journal of Experimental and Theoretical Intelligence*, 5(1-2) :1–17, 2010.

[2] U. Aickelin. An indirect genetic algorithm for set covering problems. *Journal of the Operational Research Society*, 53(10) :1118–1126, 2002.

[3] K.S. Al-Sultan, M.F. Hussain, and J.S. Nizami. A genetic algorithm for the set covering problem. *Journal of the Operational Research Society*, pages 702–709, 1996.

[4] S. Athanassopoulos, I. Caragiannis, and C. Kaklamanis. Analysis of approximation algorithms for k-set cover using factor-revealing linear programs. *Theory of Computing Systems*, 45(3) :555–576, 2009.

[5] T. Bäck, M. Schütz, and S. Khuri. A comparative study of a penalty function, a repair heuristic, and stochastic operators with the set-covering problem. In *Artificial Evolution*, pages 320–332. Springer, 1996.

[6] E. Balas. A class of location, distribution and scheduling problems : Modeling and solution methods. In *Proceedings of the Chinese-US Symposium on Systems Analysis*, pages 323–346, New York, NY, USA, 1983. Wiley.

[7] E. Balas and M.C. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44(6) :875–890, 1996.

[8] E. Balas and M.W. Padberg. Set partitioning : A survey. *Siam Review*, 18(4) :710–760, 1976.

[9] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2) :262–275, 1998.

[10] R. Bar-Yehuda. Using homogenous weights for approximating the partial cover problem. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 71–75. Society for Industrial and Applied Mathematics, 1999.

[11] R. Battiti and M. Protasi. Reactive search, a history-sensitive heuristic for max-sat. *Journal of Experimental Algorithmics (JEA)*, 2(2) :1–31, 1997.

[12] R. Battiti, G. Tecchiolli, et al. The reactive tabu search. *ORSA journal on computing*, 6 :126–126, 1994.

[13] J. Bautista and J. Pereira. A GRASP algorithm to solve the unicost set covering problem. *Computers & Operations Research*, 34(10) :3162–3173, 2007.

[14] J.E. Beasley. A lagrangian heuristic for set-covering problems. *Naval Research Logistics*, 37(1) :151–164, 1990.

[15] J.E. Beasley. OR-Library : Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11) :1069–1072, 1990.

[16] J.E. Beasley and P.C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2) :392–404, 1996.

[17] J.E. Beasley and K. Jörnsten. Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58(2) :293–300, 1992.

[18] D. Bertsimas and R. Vohra. Rounding algorithms for covering problems. *Mathematical Programming*, 80(1) :63–89, 1998.

[19] N. Bilal, P. Galinier, and F. Guibault. A new formulation of the set covering problem for metaheuristic approaches. *ISRN Operations Research*, 2013.

[20] N. Bilal, P. Galinier, and F. Guibault. An iterated-tabu-search heuristic for a variant of the partial set covering problem. *Journal of Heuristics*, 20(2) :143–164, 2014.

[21] A. Caprara, M. Fischetti, and P. Toth. A heuristic algorithm for the set covering problem. *Integer Programming and Combinatorial Optimization*, pages 72–84, 1996.

[22] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations research*, 47(5) :730–743, 1999.

[23] A. Caprara, P. Toth, and M. Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1) :353–371, 2000.

[24] M. Caserta. Tabu search-based metaheuristic algorithm for large-scale set covering problems. *Metaheuristics*, pages 43–63, 2007.

[25] S. Ceria. Set covering problem. *Annotated bibliographies in combinatorial optimization*, 1997.

[26] S. Ceria, P. Nobili, and A. Sassano. A lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81 :215–228, 1998.

[27] P. Chen, Y. Qu, H. Huang, and X. Dong. A new hybrid iterated local search for the open vehicle routing problem. In *Computational Intelligence and Industrial Application, 2008. PACIIA'08. Pacific-Asia Workshop on*, volume 1, pages 891–895. IEEE, 2008.

[28] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, pages 233–235, 1979.

[29] R.K. Congram, C.N. Potts, and S.L. Van De Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1) :52–67, 2002.

[30] D. Costa. An evolutionary tabu search algorithm and the nhl scheduling problem. *Infor-Information Systems and Operational Research*, 33(3) :161–178, 1995.

[31] B. Crawford and C. Castro. Integrating lookahead and post processing procedures with aco for solving set partitioning and covering problems. *Artificial Intelligence and Soft Computing–ICAISC 2006*, pages 1082–1090, 2006.

[32] F. Dammeyer, P. Forst, and S. Voss. Technical note on the cancellation sequence method of tabu search. *ORSA Journal on Computing*, 3(3) :262–265, 1991.

[33] F. Dammeyer and S. Voß. Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research*, 41(2) :29–46, 1993.

[34] G.W. DePuy, G.E. Whitehouse, and R.J. Moraga. Using the meta-raps approach to solve combinatorial problems. In *Proceedings of the 2002 Industrial Engineering Research Conference, May*, volume 19, page 21. Citeseer, 2002.

[35] A.V. Eremeev. A genetic algorithm with a non-binary representation for the set covering problem. In *Operations Research Proceedings 1998*, pages 175–181. Springer, 1999.

[36] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2) :109–133, 1995.

[37] M.L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, pages 1–18, 1981.

[38] M.L. Fisher and P. Kedia. Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, 36(6) :674–688, 1990.

[39] C. Fleurent and J.A. Ferland. Genetic hybrids for the quadratic assignment problem. *Quadratic assignment and related problems*, 16 :173–187, 1994.

[40] M.J. Freitas Souza, M.T. Mine, M.D.S. Alves Silva, and A. Ochi, L.S. andSubramanian. A hybrid heuristic, based on iterated local search and genius, for the vehicle routing problem with simultaneous pickup and delivery. *International Journal of Logistics Systems and Management*, 10(2) :142–157, 2011.

[41] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. In Fernando Orejas, Paul Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 225–236. Springer Berlin / Heidelberg, 2001.

[42] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman, Oxford, UK, 1979.

[43] F. Glover. Tabu search-part II. *ORSA Journal on computing*, 2(1) :4–32, 1990.

[44] F. Glover et al. Tabu search-part I. *ORSA Journal on computing*, 1(3) :190–206, 1989.

[45] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 39(3) :653–684, 2000.

[46] F. Glover and E. Taillard. A user's guide to tabu search. *Annals of operations research*, 41(1) :1–28, 1993.

[47] N.G. Hall. Multicovering problem. *European Journal of Operational Research*, 62(3) :323 – 339, 1992.

[48] F. Harche and G.L. Thompson. The column subtraction algorithm : An exact method for solving weighted set covering, packing and partitioning problems. *Computers & Operations Research*, 21(6) :689–705, 1994.

[49] M. Held, P. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical programming*, 6(1) :62–88, 1974.

[50] T. Jiang, Q. Cui, G. Shi, and S. Ma. Protein folding simulations of the hydrophobic–hydrophilic model by combining tabu search with genetic algorithms. *The Journal of chemical physics*, 119 :4592, 2003.

[51] T. Jiang and F. Yang. An evolutionary tabu search for cell image segmentation. *Systems, Man, and Cybernetics, Part B : Cybernetics, IEEE Transactions on*, 32(5) :675–678, 2002.

[52] D.S. Johnson and L.A. McGeoch. The traveling salesman problem : A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local search in combinatorial optimization*, volume 1, pages 215–310. John Wiley and Sons, Ltd., 1997.

[53] K. Katayama, H. Narihisa, et al. Iterated local search approach using genetic transformation to the traveling salesman problem. In *Proc. of GECCO 99*, volume 1, pages 321–328, 1999.

[54] S. Khuller, A. Moss, and J.S. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1) :39–45, 1999.

[55] G. Kinney, JW Barnes, and B. Colleti. A group theoretic tabu search algorithm for set covering problems. Technical report, Working paper, available from http ://www. me. utexas. edu/barnes/research, 2004.

[56] G.W. Kinney, J.W. Barnes, and B.W. Colletti. A reactive tabu search algorithm with variable clustering for the unicost set covering problem. *International Journal of Operational Research*, 2(2) :156–172, 2007.

[57] J. Könemann, O. Parekh, and D. Segev. A unified approach to approximating partial covering problems. In *Algorithms–ESA 2006*, pages 468–479. Springer, 2006.

[58] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3) :125–138, 2000.

[59] S. Kurahashi and T. Terano. A genetic algorithm with tabu search for multimodal and multiobjective function optimization. In *Proc. the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 291–298, 2000.

[60] G. Lan. An effective and simple heuristic based on meta-raps to solve large-scale set covering problems. Master's thesis, University of Louisville, 2004.

[61] G. Lan, G.W. DePuy, and G.E. Whitehouse. An effective and simple heuristic for the set covering problem. *European journal of operational research*, 176(3) :1387–1403, 2007.

[62] Y. Lee, H.D. Sherali, I. Kwon, and S. Kim. A new reformulation approach for the generalized partial covering problem. *Naval Research Logistics (NRL)*, 53(2) :170–179, 2006.

[63] L. Lessing, I. Dumitrescu, and T. Stützle. A comparison between aco algorithms for the set covering problem. *Ant Colony Optimization and Swarm Intelligence*, pages 105–122, 2004.

[64] H.R. Lourenço, O.C. Martin, and T. Stützle. Iterated local search. In *Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science*, pages 321–353. Kluwer Academic Publishers, 2002.

[65] H.R. Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu-search : Application to the job-shop scheduling problem. In *Meta-Heuristics*, pages 219–236. Springer, 1996.

[66] O. Martin, S.W. Otto, and E.W. Felten. *Large-step Markov chains for the traveling salesman problem.* Technical report. Oregon Graduate Institute of Science and Technology, Dept. of Computer Science and Engineering, 1991.

[67] O.C. Martin and S.W. Otto. Partitioning of unstructured meshes for load balancing. *Concurrency : Practice and Experience*, 7(4) :303–314, 1995.

[68] O.C. Martin and S.W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63(1) :57–75, 1996.

[69] V. Melkonian. New primal–dual algorithms for steiner tree problems. *Computers & operations research*, 34(7) :2147–2167, 2007.

[70] A. Misevičius. Using iterated tabu search for the travelling salesman problem. *Information technology and control*, 32(3) :29–40, 2004.

[71] A. Misevičius, A. Lenkevicius, and D. Rubliauskas. Iterated tabu search : an improvement to standard tabu search. *Information Technology and Control*, 35(3) :187–197, 2006.

[72] N. Musliu. Local search algorithm for unicost set covering problem. *Advances in Applied Artificial Intelligence*, 4031 :302–311, 2006.

[73] Z. Naji-Azimi, P. Toth, and L. Galli. An electromagnetism metaheuristic for the unicost set covering problem. *European Journal of Operational Research*, 205(2) :290–300, 2010.

[74] D. Orvosh and L. Davis. Using a genetic algorithm to optimize problems with feasibility constraints. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 548–553. IEEE, 1994.

[75] G. Palubeckis. Iterated tabu search for the maximum diversity problem. *Applied mathematics and computation*, 189(1) :371–383, 2007.

[76] G. Pan. Geostatistical design of infill drilling programs. *Society of Mining Engineers of AIME*, 142, 1995.

[77] N.J. Radcliffe and P.D. Surry. Formal memetic algorithms. In *Evolutionary Computing*, pages 1–16. Springer, 1994.

[78] M. Rahoual, R. Hadji, and V. Bachelet. Parallel ant system for the set covering problem. *Lecture Notes in Computer Science*, 2463 :262–267, 2002.

[79] Z. Ren, Z. Feng, L. Ke, and H. Chang. A fast and efficient ant colony optimization approach for the set covering problem. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1839–1844. IEEE, 2008.

[80] Z. Ren, Z. Feng, L. Ke, and H. Chang. A fast and efficient ant colony optimization approach for the set covering problem. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1839–1844. IEEE, 2008.

[81] Z.G. Ren, Z.R. Feng, L.J. Ke, and Z.J. Zhang. New ideas for applying ant colony optimization to the set covering problem. *Computers & Industrial Engineering*, 58(4) :774–784, 2010.

[82] K. Saikia and B. C. Sarkar. Exploration drilling optimisation using geostatistics : a case in jharia coalfield, india. *Applied Earth Science*, 115(1) :13–22, 2006-04-01T00 :00 :00.

[83] K. Smyth, H.H. Hoos, and T. Stützle. Iterated robust tabu search for max-sat. In *Advances in Artificial Intelligence*, volume 2671 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2003.

[84] M. Solar, V. Parada, and R. Urrutia. A parallel genetic algorithm to solve the set-covering problem. *Computers & Operations Research*, 29(9) :1221–1235, 2002.

[85] S. Soltani and A. Hezarkhani. Proposed algorithm for optimization of directional additional exploratory drill holes and computer coding. *Arabian Journal of Geosciences*, 6(2) :455–462, 2013.

[86] S. Soltani, A. Hezarkhani, A. Erhan Tercan, and B. Karimi. Use of genetic algorithm in optimally locating additional drill holes. *Journal of Mining Science*, 47 :62–72, 2011. 10.1134/S1062739147010084.

[87] T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical report, FG Intellektik, TU Darmstadt, Darmstadt, Germany, 1998.

[88] R. Thamilselvan and D.P. Balasubramanie. Integrating genetic algorithm, tabu search approach for job shop scheduling. *Arxiv preprint arXiv :0906.5070*, 2009.

[89] R. Thamilselvan and P. Balasubramanie. A genetic algorithm with a tabu search (gta) for traveling salesman problem. *International Journal of Recent Trends in Engineering.*, 1 :607–610, 2009.

[90] R.L. Wang and K. Okazaki. An improved genetic algorithm with conditional genetic operators and its application to set-covering problem. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 11(7) :687–694, 2007.

[91] wikipedia. Krigeage. http ://fr.wikipedia.org/wiki/Krigeage.

[92] M. Yagiura, M. Kishida, and T. Ibaraki. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, 172(2) :472 – 499, 2006.

[93] B. Yelbay, S.I. Birbil, and K. Bülbül. The set covering problem revisited : An empirical study of the value of dual information. *optimization-online. "http ://www.optimization-online.org/DB_HTML/2010/11/2814.html"*, 2012.

[94] R. Zanjirani Farahani and M. Hekmatfar. *Facility Location*, chapter 7.3. Springer Dordrecht Heidelberg London New York, 2009.

[95] G. Zäpfel, R. Braune, and M. Bögl. *Metaheuristic Search Concepts : A Tutorial with Applications to Production and Logistics*. Springer, 2010.

[96] M. Zdánskỳ and J. Pozivil. Combination genetic/tabu search algorithm for hybrid flowshops optimization. In *Proceedings of ALGORITMY*, pages 230–236, 2002.