Titre: Title:	Étude et programmation de l'algorithme du flot maximum appliqué aux problèmes de contours ultimes dans une mine à ciel ouvert
Auteur: Author:	Geneviève Auger
Date:	2000
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Auger, G. (2000). Étude et programmation de l'algorithme du flot maximum appliqué aux problèmes de contours ultimes dans une mine à ciel ouvert [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8855/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8855/
Directeurs de recherche: Advisors:	Michel Gamache
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

ÉTUDE ET PROGRAMMATION DE L'ALGORITHME DU FLOT MAXIMUM APPLIQUÉ AUX PROBLÈMES DE CONTOURS ULTIMES DANS UNE MINE À CIEL OUVERT

GENEVIÈVE AUGER
DÉPARTEMENTS DES GÉNIES CIVIL, GÉOLOGIQUE ET MINES
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE MINÉRAL)
DÉCEMBRE 2000

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

ÉTUDE ET PROGRAMMATION DE L'ALGORITHME DU FLOT MAXIMUM APPLIQUÉ AUX PROBLÈMES DE CONTOURS ULTIMES DANS UNE MINE À CIEL OUVERT

présenté par : AUGER Geneviève

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. MARCOTTE Denis, Ph.D., président

M. GAMACHE Michel, Ph.D., membre et directeur de recherche

M. OUELLET Jacques, Ph.D., membre

J'aimerais dédier cet ouvrage à la
mémoire de mes chères disparues,
Irène et Francine.

REMERCIEMENTS

Merci tout d'abord au CRSNG pour m'avoir permis de faire ces études et aussi pour m'avoir accordé un congé de maternité. Merci à la mine «M», pour nous avoir fourni si gracieusement un fichier représentant un de leur gisement avec des données réelles. Merci à Martine Bellaiche et à Pierre Girard pour m'avoir guidée dans les dédales de la programmation alors que j'étais tout à fait désespérée devant le programme de flot maximum quant à la façon de faire fonctionner mon programme sur une station UNIX. Merci à Lucette De Gagné pour m'avoir aidée à plusieurs reprises à régler quelques petits problèmes de nature informatique. Merci à Chantal Tellier, une grande amie, qui est toujours là lorsqu'on a besoin d'elle et qui a corrigé le français de mon mémoire. Merci à Daniel Bulota pour m'avoir aidée à traduire l'abstract. Merci à Louis-Philippe Bigras pour m'avoir aidée au tout début à comprendre de quelle façon le logiciel SURPAC fonctionnait et pour m'avoir aidée à jouer avec le très gros fichier qui nous avait été envoyé par la mine «M». Merci aux amis du local : Christiane, Martin, Renaud, Ghislain, Christian, Anne-Marie, Anna, Catherine, Vincent et Mathieu qui ont fait que ces deux années passées dans un petit «coqueron» ne m'ont pas paru trop longues. Merci à Frédéric, pour m'avoir dit à maintes reprises qu'il était fier de moi et merci à François pour être ce qu'il est, un bijou de petit garçon. Merci à mes parents, Louise et Rosaire, pour m'avoir encouragée depuis que je suis toute petite à étudier et à me surpasser. C'est un peu à cause d'eux que j'en suis maintenant là.

Et, finalement, le plus gros des mercis à Michel Gamache parce qu'il est le meilleur directeur de recherche qu'un étudiant puisse rêver d'avoir. Merci à lui pour m'avoir fait confiance à plusieurs reprises, pour m'avoir offert des occasions et pour avoir été si présent. Il a toujours été très disponible, compréhensif, intéressé, jovial et de bon conseil. Il s'est assis des après-midi complets avec moi, alors qu'il était lui-même très occupé, afin de «déboguer» le programme.

RÉSUMÉ

La planification à long terme dans les mines à ciel ouvert se traduit par la détermination des contours ultimes et par la détermination d'une séquence de production pour parvenir à réaliser ces contours ultimes. Ces deux problèmes sont célèbres en recherche opérationnelle de par la difficulté de résolution qu'ils offrent. Chacun d'eux peut être résolu à l'aide d'un flot maximum dans un graphe qui représente le gisement.

Ce mémoire de maîtrise porte sur la planification à long et à moyen terme dans les mines à ciel ouvert. Plus précisément, le but est de fournir un outil efficace, ou qui offre un bon potentiel, qui pourrait calculer des contours ultimes d'une fosse et, éventuellement, des séquences de production.

La nature du problème de la planification dans les mines à ciel ouvert ainsi que les paramètres, les définitions et la nomenclature utilisés dans la littérature y sont présentés. Une revue exhaustive de la littérature permet d'établir les méthodes qui ont été développées et employées jusqu'à maintenant, autant pour calculer les contours ultimes que les séquences de production. Puis, nous présentons la théorie qui concerne la méthode du flot maximum appliquée au problème des contours ultimes et des séquences de production que nous jugeons être la plus apte à fournir de bonnes solutions dans un temps raisonnable. Avec l'analyse des algorithmes du flot maximum, on se rend compte qu'il est inutile d'utiliser des techniques de pré-traitement qui réduisent la grosseur du graphe parce que l'algorithme du flot maximum considère implicitement ce genre de réduction durant l'exécution de l'algorithme.

Cette méthode est ensuite programmée afin de pouvoir effectuer des tests sur sa rapidité d'exécution à l'aide de données réelles fournies par une mine. Les résultats des tests montrent que les temps de calcul varient de façon linéaire en fonction du nombre de blocs. De plus, à l'aide de tests similaires, il est montré que les temps de calcul varient

aussi de façon linéaire en fonction du nombre de niveaux. Ces tests effectués avec l'algorithme du flot maximum ont montré que l'utilisation de cet algorithme appliqué au problème des contours ultimes ne représente sûrement pas le pire cas en terme de temps de résolution par rapport au comportement qui pourrait être observé selon l'étude de complexité de l'algorithme.

Finalement, une dernière stratégie, qui consiste à inverser les arcs du graphe lorsque le ratio de la valeur de la coupe au nœud source sur la valeur de la coupe au nœud puits est supérieur à 1, est utilisée. Cette stratégie s'avère la meilleure amélioration possible pour accélérer le temps de résolution du flot maximum.

ABSTRACT

Long term planning of open pit mines is accomplished by determining the ultimate contour limits as well as the production sequences that will yield the ultimate contour limits. These problems are notorious in operational research due to challenges they present. Each can be resolved by maximum flow algorithm based on a graph representing the deposit.

This master's thesis is about long-term and mid-term planning of open pit mines. Explicitly, the goal is to develop an effective tool, or at least one that offers a good potential, that could calculate the ultimate contour limits of an open pit mine and eventually the production sequences that will yield the ultimate contour limits.

The thesis will present the problems encountered in open pit mine planning as well as the parameters, the definitions and the nomenclatures used in literature. A complete review of the literature reveals the methods that were developed and used up until now to figure out the ultimate contour limits and the production sequences. Following this review, the theory surrounding the maximum flow method applied to the ultimate contour limits and production sequence problem, which we find capable of delivering good results in a timely matter, will be presented. The analysis of the maximum flow algorithm shows that it is futile to use pre-processing techniques aimed at reducing the size of the graph since the maximum flow algorithm implicitly considers this type of reduction during its execution.

This method is then programmed in a way that will test its own speed of execution based on real-time data supplied by a mine. The results show that processing time varies in a linear fashion with respect to the number of blocs. Also with similar tests, it is shown that the processing time varies in a linear fashion with respect to the number of levels. These tests done on the maximum flow algorithm reveal that when the algorithm is

applied to the ultimate contour limits problem the time required is definitely no worse than the behavior observed with respect to the study of complexity of the algorithm.

Finally, another strategy is used and consists in reversing the arcs of the graph when the ratio between the value of the source node and the value of the sink node is greater than 1. This strategy is the best improvement one can do in order to decrease the processing time of the maximum flow algorithm.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

CHAPITRE I
INTRODUCTION

L'industrie minière en est une d'importance au Canada où l'activité liée aux ressources naturelles représentent 13,1% du PIB selon les données de Statistique Canada en 1997. Non seulement elle a de l'importance sur le plan de l'économie, mais aussi sur le plan technologique. En effet, dans une ère où les gisements se font de plus en plus rares et de moins en moins riches, il devient capital de développer de nouvelles technologies qui permettront à l'industrie minière de survivre et de continuer à prospérer.

Qu'elles soient souterraines ou à ciel ouvert, les exploitations minières présentent des défis de taille en recherche opérationnelle et en optimisation des opérations. Les techniques de recherche opérationnelle peuvent aider à résoudre plusieurs problèmes rencontrés dans l'industrie. On pense entre autres aux problèmes de plus courts chemins, d'utilisation de l'équipement, de mélange, d'affectation (pelles et camions), de localisation (concasseur, puits, concentrateur), de capacité, de files d'attente, de «dispatching», de transport, de sélection de machinerie et de planification.

Entre tous ces problèmes, la planification des opérations minières à ciel ouvert représente un défi de taille particulièrement attrayant étant donné la complexité de résolution d'un tel problème considérant le nombre de contraintes et de variables qu'il implique. Une mauvaise planification lors de l'extraction du minerai peut entraîner des difficultés importantes. Presque tout dans une exploitation minière est fait en fonction du profit à réaliser. Des pertes de temps occasionnées par une mauvaise planification peuvent engendrer une perte de profit considérable. Une mauvaise planification peut mener à une mauvaise récupération du minerai, qu'il faudra retourner chercher ultérieurement ou, pire encore, qu'il faudra laisser en place. Une mauvaise planification peut avoir aussi pour conséquence l'obligation de déménager certaines infrastructures

telles que des routes ou des bâtiments. Ce sont donc toutes de bonnes raisons pour tenter de trouver un moyen efficace et optimal de planifier les opérations minières.

Actuellement, il n'existe pas une grande variété de logiciels sur le marché qui aident les gens de l'industrie minière lors de la planification à long et à moyen terme de leur exploitation à ciel ouvert. Le plus populaire d'entre eux exige parfois plus de deux jours de travail pour faire les calculs d'un gisement d'environ 700 000 blocs. Il y a donc place à l'amélioration et c'est pourquoi ce mémoire se penche sur une partie du problème de la planification dans les mines à ciel ouvert.

Dans cet ouvrage, nous allons donc traiter de la planification à long et à moyen terme dans les mines à ciel ouvert. Le but est de fournir à l'industrie minière un outil efficace et flexible qui peut calculer des contours ultimes d'une fosse et, éventuellement, établir les séquences de production. Un autre but est de montrer que cet outil pourra se greffer aux outils de gestion déjà en utilisation dans les mines.

Dans le chapitre II, la nature du problème de la planification dans les mines à ciel ouvert ainsi que les paramètres, les définitions et la nomenclature qui seront utilisés dans ce mémoire sont présentés en détail. Une revue exhaustive de la littérature portant sur les méthodes qui ont été développées et employées jusqu'à maintenant, autant pour calculer les contours ultimes que les séquences de production est présentée au chapitre III. De cette revue, il ressort que les méthodes de résolution basées sur les algorithmes de flot maximum sont celles qui offrent le meilleur potentiel de réussite, c'est-à-dire que ces algorithmes sont les plus aptes à fournir de bonnes solutions dans un temps raisonnable. Au chapitre IV sont présentées la description et la théorie portant sur les algorithmes de flot maximum. Dans ce chapitre, les améliorations possibles de ces algorithmes seront également cernées et permettront d'identifier la meilleure implantation. Le chapitre V présente les données qui ont été utilisées pour les tests ainsi que le programme qui a été créé pour résoudre le problème des contours ultimes. Le chapitre VI présente les résultats des tests effectués pour tester la rapidité d'exécution de la méthode ainsi que

l'analyse de ces résultats. Finalement, la conclusion de l'analyse de la méthode développée et du choix des algorithmes constitue le chapitre VII. De plus dans ce chapitre, il est question des domaines de recherche qu'il faudrait aborder afin d'améliorer et d'adapter les algorithmes de flot maximum à la structure unique des problèmes de contours ultimes des mines à ciel ouvert. L'emphase est mise sur les possibilités offertes par l'outil développé pour la résolution du problème et les ajustements qui pourront être faits par la suite de façon à rendre le programme plus complet.

CHAPITRE II
PRÉSENTATION DU PROBLÈME

Dans ce chapitre, nous présentons une description détaillée du problème de planification dans les mines à ciel ouvert : les objectifs ainsi que les paramètres qui entrent en jeu. Nous y présentons aussi des définitions de base qui seront nécessaires à la bonne compréhension du mémoire et des notions qui seront présentées dans les chapitres subséquents.

2.1 Définitions : planification à long, moyen et court terme

La *planification des opérations* dans une exploitation à ciel ouvert est définie comme étant le développement d'une séquence d'exploitation qui conduit de l'état initial du gisement, c'est-à-dire de la topographie originale, vers son état final, soit les contours ultimes de la fosse. La planification des opérations peut s'effectuer en trois étapes qui dépendent de l'horizon temporel considéré. Pour la *planification à long terme*, la durée de vie économique de la mine détermine l'horizon temporel. Pour ce type de planification, on cherche à déterminer les séquences d'extraction en années. Pour la *planification à moyen terme*, l'horizon temporel est une année et les séquences d'extraction sont regroupées par périodes mensuelles ou trimestrielles. Finalement, la *planification à court terme* a un horizon temporel mensuel et indique la séquence des opérations pour chaque semaine, chaque journée ou quart de travail. Pour chacun des horizons temporels, les séquences d'extraction sont déterminées en fonction d'un objectif et doivent respecter des contraintes de différentes natures.

Les objectifs de la planification à long terme visent à estimer les réserves de minerai, déterminer les contours ultimes de la fosse et élaborer une stratégie de minage basée sur l'investissement à long terme. Les objectifs de la planification à moyen terme visent à élaborer le «design» des routes de halage, prévoir l'investissement des équipements et

déterminer la séquence de minage des réserves accessibles pour une année. Les objectifs de la planification à court terme se résument à établir la séquence des tirs de dynamitage et de forage, à prévoir les déplacements et la priorité des pelles et ce, sur une base journalière ou hebdomadaire. Toutefois, le principal objectif de toute planification dans une mine à ciel ouvert est de maximiser les profits réalisés pour chacune des périodes tout en maximisant les profits totaux de l'exploitation de la mine. Selon Fytas et al. (1993), ceci peut être exprimé comme une maximisation du taux de rendement interne, une minimisation de la période de remboursement (payback) ou par une maximisation du ratio de la valeur présente nette.

En utilisant les techniques de résolution développées en recherche opérationnelle, il est possible de maximiser les profits et d'atteindre l'objectif fixé à chacune de ces étapes. Toutefois, il est difficile de concilier le long, le moyen et le court terme.

2.2 Les pentes et les routes de halage

Plusieurs paramètres peuvent affecter la grosseur et la forme que prendra une fosse mais les plus importants sont probablement les pentes et les routes de halage. Ces dernières peuvent changer considérablement les résultats si elles ne sont pas prises en considération lors des calculs des contours ultimes ou des séquences de production.

Les pentes de la fosse déterminent quelle quantité de stérile doit être enlevée afin d'aller chercher le minerai. Elles sont généralement exprimées en degrés à partir du plan horizontal.

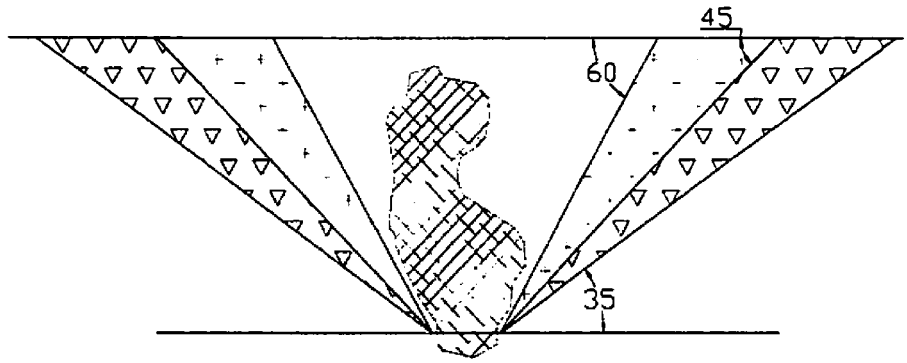
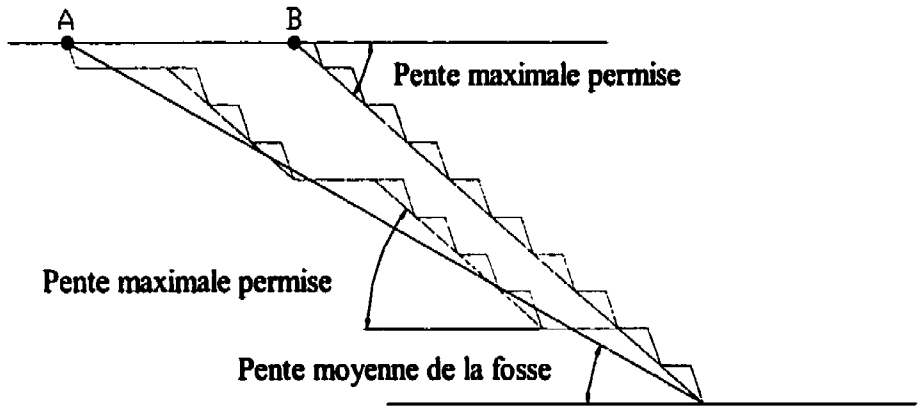


Figure 2.1 : L'importance des pentes

Les pentes doivent demeurer stables tout au long de la vie de la mine. La stabilité des pentes doit être analysée attentivement : il en va de la sécurité des équipements et du personnel. La force de la roche, les fractures, les joints, la présence d'eau, et toutes autres informations géologiques vont servir à les déterminer. Il peut y avoir une même pente pour tous les murs de la mine, ou encore des pentes différentes selon le type de roche rencontrée dans la mine. Cette dernière façon de procéder est plus réaliste. Comme l'objectif est de minimiser la quantité de stérile qui sera extrait, et s'il existe des secteurs de la mine permettant d'avoir des pentes plus abruptes, il serait pertinent d'en profiter.

Le «design» des routes est lui aussi un facteur très important. Une route de halage occupe un espace non négligeable dans une mine à ciel ouvert, surtout dans le cas des petites exploitations. Lors du «design»d'une route, une pente maximale et une certaine largeur doivent être respectées. Ces deux éléments sont fonction du type d'équipement qui est utilisé. Or, pour atteindre le fond de la fosse, une certaine longueur de route, qui dépend de la profondeur et du degré de descente désiré, est nécessaire. Généralement, il faut adoucir légèrement les pentes de la fosse pour permettre l'installation des routes de halage, ce qui signifie que beaucoup plus de stérile devra être enlevé.



A Intersection entre la surface et la fosse si les routes sont incluses dans le « design ».

B Intersection entre la surface et la fosse si on ne considère pas les routes dans le « design ».

Figure 2.2: Vue de profil d'un mur de mine avec une route et impact sur les pentes

2.3 Modélisation du gisement

La première étape dans la résolution du problème d'optimisation de la planification est de trouver un moyen de modéliser le gisement en question. Toutes les méthodes développées jusqu'à présent, à l'exception des algorithmes génétiques de Schofield et Denby (1993), utilisent un modèle de blocs tridimensionnels, généralement réguliers, pour représenter le gisement. Cette façon de modéliser le gisement fut développée initialement pour déterminer les contours ultimes d'une fosse. Toutefois, dans la détermination des séquences de minage, que ce soit à long, moyen ou court terme, l'idée est conservée et utilisée comme base de travail.

2.3.1 La division du gisement en blocs

L'étape de modélisation du gisement est très importante dans le processus du «design» assisté par ordinateur car le résultat du «design» optimal repose essentiellement sur cette

division artificielle qui a été faite du gisement. Donc, la modélisation se doit d'être représentative et la plus juste possible.

Étant donné que le traitement de données sous forme de matrice convient parfaitement aux ordinateurs, on décide dans la plupart des cas de subdiviser le gisement en blocs de même dimension ou de dimensions variables. Seulement quelques auteurs (Lemieux 1977, Lee et Kim 1979, Braticevic 1984 et Stuart 1992) ont proposé des modèles de blocs où les dimensions étaient variables et ce, dans le but de pouvoir modéliser des gisements très irréguliers ou encore pour réduire la grosseur du modèle. Toutefois, la façon conventionnelle de diviser un gisement est d'utiliser des blocs cubiques ou rectangulaires de même dimension. La plupart des articles qui traitent d'études de cas ou qui donnent une revue de la littérature décrivent très bien de quelle façon il faut procéder pour modéliser le gisement. Parmi ces articles, on retrouve Gauthier et Gray (1971), Marino et Slama (1972), Phillips (1973), David et al. (1974), Kim (1978, 1979), Lizotte (1988), Dowd et Onur (1993), Mastoris et Topuz (1995), Baafi et al. (1995), Wheeler (1997), etc. Chacun des blocs est identifié par une adresse qui utilise le système cartésien en trois dimensions comme base, c'est-à-dire trois axes orthogonaux: x, y et z. La géométrie du gisement et l'algorithme employé vont généralement décider du type de modèle de blocs sera employé (Lizotte 1988).

La construction d'un modèle de blocs implique qu'une méthode d'extrapolation et d'interprétation des carottes de forage existe pour transposer l'information sur chacun des blocs. Selon Lizotte (1988), la seule vraie méthode pour déterminer les teneurs sur chacun des blocs est le krigeage, une méthode géostatistique. Cette méthode fournit non seulement la teneur d'un bloc mais, de plus, spécifie la confiance associée à cet estimé. À partir de cette information, il sera possible de prendre la décision d'extraire le bloc comme minerai, de l'extraire comme stérile ou de le laisser en place. François-Bongarçon et Maréchal (1977) et Coléou (1988) soutiennent qu'il s'agit là d'une erreur et qu'il est impossible d'attribuer des valeurs à de petits blocs et que seuls de grands panneaux peuvent être représentatifs des informations disponibles lors des études de

détermination des contours ultimes. Quoiqu'il en soit, la majorité des méthodes qui ont été développées jusqu'à maintenant ont pour base un gisement divisé en blocs de même grosseur.

2.3.2 La teneur de coupure

Avant d'approfondir les explications concernant la modélisation du gisement, il serait préférable d'introduire la notion de teneur de coupure. Toutes les exploitations, qu'elles soient à ciel ouvert ou souterraines, utilisent cette notion et il est important de bien la définir. Les meilleures définitions concernant les différents types de teneurs de coupure sont présentées par Taylor (1972) et Armstrong (1990). Lors des opérations, on doit prendre une décision sur ce qui va advenir du prochain bloc. Sera-t-il extrait et traité ? extrait et entreposé ? seulement extrait (stérile) et envoyé sur les haldes à stérile ou laissé en place ? C'est la teneur du bloc qui est considérée pour prendre cette décision.

Tout d'abord, la *teneur* en métal d'un minerai représente la quantité relative de substance utile que contient ce minerai. Cette proportion s'exprime de différentes façons selon les minerais. Ainsi on parlera, par exemple, d'un minerai de cuivre de 2%, d'un minerai d'or de 0,5 once la tonne ou d'un minerai d'amiante de 7,00$ la tonne.

La *teneur de coupure* est une teneur choisie et utilisée afin de démarquer deux lignes de conduite : soit exploiter un gisement ou le laisser en place, soit traiter le minerai ou le rejeter. Ce peut être également une suite de teneurs utilisées pour tronquer une fonction de fréquence de distribution ou pour séparer du matériel minéralisé en fractions graduées.

La *teneur de coupure optimale* est la teneur de coupure, qui par le choix qu'on en fait, permet de maximiser le bénéfice.

La *teneur limite* est la teneur à laquelle le revenu récupérable du minerai est égal aux coûts d'exploitation, de concentration et de mise en marché du minerai.

Les teneurs de coupure sont utilisées pour définir la portion du matériel qui sera extraite de la mine et traitée au concentrateur. Il existe deux types de teneur de coupure, soit la teneur de coupure de planification et la teneur de coupure d'opération.

La *teneur de coupure de planification* est utilisée au cours de la phase d'exploration et aux différentes étapes de la planification de l'exploitation d'un gisement. Cette teneur de coupure est nécessaire pour définir géographiquement et quantitativement les limites du minerai potentiel. Puisqu'aucune information précise sur l'exploitation minière éventuelle n'est encore connue au début de l'exploration d'un site, la teneur de coupure de planification à cette étape est semi-quantitative, c'est-à-dire qu'elle vise à prédire de façon satisfaisante la valeur de minerai qui pourra éventuellement être exploité ou traité. En général, les réserves sont calculées par « enveloppes » pour une gamme de teneurs de coupure. Certaines parties sous la teneur de coupure peuvent être incluses pour des raisons techniques, c'est-à-dire la méthode d'exploitation utilisée.

La *teneur de coupure d'opération* est nécessaire au début de l'exploitation pour déterminer à court terme quelle portion du minerai peut être gardée en réserve et quelle portion peut être acheminée vers le concentrateur. Il existe trois types de teneur de coupure d'opération, soit la teneur de coupure à la mine, la teneur de coupure au concentrateur et, finalement, la teneur de coupure à la fonderie (plus rare).

La *teneur de coupure à la mine* signifie que le minerai au-dessus de la teneur de coupure sera exploité et ce,lui en dessous restera en place. Un bloc qui peut couvrir les coûts de son extraction, de son traitement et de sa mise en marché est un bloc dont la teneur est supérieure ou égale à la teneur de coupure à la mine. Toutefois, plusieurs blocs en dessous de la teneur de coupure doivent être extraits afin de permettre aux blocs rentables d'être extraits et parmi eux, quelque uns ont une certaine valeur.

La *teneur de coupure au concentrateur* signifie que les blocs qui ont une teneur supérieure à la teneur de coupure sont acheminés vers l'usine de traitement pour être concentrés et que les blocs sous la teneur de coupure sont jetés comme stérile ou

entreposés sur des haldes à minerai (faible teneur). Cette teneur de coupure est généralement utilisée pour les blocs qui n'auraient pas été extraits pour leur propre valeur, c'est-à-dire ceux qui sont en-dessous de la teneur de coupure à la mine. Ils ont été extraits comme du stérile, donc leur coût d'extraction a déjà été pris en charge par un autre bloc de minerai. La destination finale de ce bloc n'est influencée à présent que par le coût de traitement et de mise en marché. Donc un bloc qui peut couvrir ces frais a une teneur supérieure ou égale à la teneur de coupure au concentrateur.

Les blocs qui ont été entreposés sur des haldes à minerai sont des blocs qui possèdent une très faible teneur et qui ne peuvent pas être traités pour le moment. Ces réserves de minerai à faible teneur pourront être traitées lorsque le prix du minerai sur le marché montera ou encore pour diluer les trop fortes teneurs au concentrateur.

Le concept de teneur de coupure étant maintenant bien défini, nous pouvons revenir au concept de modélisation du gisement. Une fois que la teneur de coupure à la mine est déterminée, chacun des blocs du gisement est identifié comme étant un bloc de stérile ou un bloc de minerai. Lors de l'extraction du minerai, on tentera le plus possible de laisser les blocs de stérile en place et d'aller chercher le maximum des blocs de minerai. Selon la teneur de coupure au concentrateur, les blocs de minerai iront soit au concentrateur, soit sur des haldes de minerai à faible teneur. Si un bloc de stérile doit être extrait absolument, ce dernier ira directement sur des haldes à stérile. Chaque fois qu'un bloc est extrait, que ce soit du minerai ou du stérile, cela implique des coûts d'extraction et de transport. Si un bloc de minerai est envoyé au concentrateur, il implique des coûts de traitement. S'il est envoyé sur une halde de minerai pour subir un traitement futur, il impliquera des coûts supplémentaires de transport pour la manutention des matériaux de la halde vers l'usine de traitement en plus des coûts de traitement.

2.3.3 La grosseur des blocs

La grosseur des blocs doit être reliée à l'unité de minage, c'est-à-dire la taille la plus petite avec laquelle il est possible de décider si on extrait ou pas le bloc. La pratique

courante est d'avoir des blocs de hauteur égale à celle des bancs. Selon Lizotte (1988), pour les autres dimensions, largeur et profondeur, il est souvent plus court, en termes de temps de calcul, d'avoir des blocs cubiques ou qui représentent la pente des talus. Le bloc peut parfois être plus gros si le contrôle des teneurs à une échelle plus petite n'est pas possible, par exemple, dans le cas où la méthode de minage utilisée nécessiterait de gros dynamitages. Dans de rares circonstances, l'unité de minage pourrait être plus petite. Cette occasion se présente dans le cas des mines d'uranium, par exemple, où des mesures de la radioactivité peuvent permettre des décisions sur des blocs de la grosseur du chargement d'un camion.

2.3.4 L'information fournie sur chacun des blocs

Il faut rappeler que le but premier de la création d'un modèle du gisement en blocs est de pouvoir construire le «design» d'une fosse. Donc chacun des blocs doit pouvoir contenir plusieurs informations nécessaires aux calculs de ce «design» sous une forme qui pourra être traitée par un ordinateur.

Pour chacun de ces blocs, on doit donc connaître un estimé de la teneur, du tonnage, de la proportion de chacun des minerais présents et des impuretés qui pourraient affecter l'efficacité du traitement de concentration. Avec ces estimations, on peut établir des caractéristiques minéralogiques et métallurgiques qui permettront à leur tour d'estimer le pourcentage de récupération du minerai, les revenus et les coûts de traitement de chaque bloc de minerai. En y ajoutant les informations sur la position du bloc dans le gisement, on peut estimer les coûts d'extraction et de transport pour tous les blocs du gisement. À l'aide de toutes ces données, une valeur est attribuée à chacun des blocs. Cette valeur représente une estimation du profit ou de la perte qui est engendrée pour extraire ce bloc. Généralement la valeur est positive lorsqu'il s'agit de minerai et négative pour du stérile ou un bloc de minerai en dessous de la teneur de coupure à la mine. Ce qu'il est important de toujours garder en mémoire lorsqu'on utilise une modélisation d'un

gisement, c'est que la valeur du bloc repose sur une estimation plus ou moins précise de sa teneur dépendant de la méthode utilisée et de la qualité de l'information fournie.

CHAPITRE III
REVUE DE LA LITTÉRATURE

Ce chapitre qui traite de la revue de la littérature sera divisé en deux grandes parties. Dans la première partie, nous allons parler des techniques qui ont été développées au cours des ans afin de déterminer les contours ultimes d'une exploitation à ciel ouvert. La deuxième partie, quant à elle, portera sur la détermination des séquences optimales de production.

3.1 Contours ultimes

Il n'y a pas si longtemps que la recherche opérationnelle est utilisée pour résoudre les problèmes de planification des opérations dans les mines à ciel ouvert. L'un des premiers problèmes qu'on tenta de résoudre dans ce domaine fut la détermination des contours ultimes d'une exploitation en fosse. Les contours ultimes d'une mine à ciel ouvert représentent les contours géométriques de la mine après son exploitation ou encore ils représentent l'allure de la mine à la fin de sa vie économique. Les contours optimaux sont ceux qui permettent de maximiser les profits totaux sans tenir compte du temps. Il s'agit d'un problème de planification à long terme.

La détermination des contours finaux d'une mine à ciel ouvert est essentielle pour une planification adéquate. Elle permet entre autres d'estimer la quantité de minerai qu'il sera possible d'extraire du gisement, d'estimer la durée économique de l'exploitation, de planifier la dimension des installations de surface et de la machinerie et de planifier la capacité de production. La connaissance des contours permet également de planifier et d'organiser les séquences d'exploitation à court, moyen et long terme qui permettront de maximiser les profits.

Plusieurs personnes ont tenté de concevoir un algorithme qui permettrait d'optimiser les contours finaux d'une mine à ciel ouvert. La tâche la plus difficile est de développer un

algorithme qui soit facile à implanter dans l'industrie, qui repose sur un modèle réaliste de la mine et dont les solutions pourront être appliquées aisément. Les articles de Kim (1978), Lizotte (1988), Dowd et Onur (1993) et Hochbaum et Chen (1998) fournissent d'excellents aperçus des techniques développées jusqu'à maintenant dans le domaine de la détermination des contours ultimes.

Les méthodes heuristiques ont été les premières méthodes développées pour déterminer les contours finaux des exploitations en fosse. Une méthode heuristique est une méthode approximative qui permet de résoudre souvent rapidement des problèmes, mais qui toutefois ne garantit pas toujours l'obtention d'une solution optimale. Parmi les heuristiques développées pour le problème des contours ultimes, celles qui utilisent la méthode des cônes mobiles ou, cônes flottants, sont les plus connues et ont été les plus utilisées dans l'industrie à cause de leur simplicité tant sur le plan de la compréhension que sur le plan de l'implantation. C'est Pana, en 1965, qui a le premier proposé cette méthode et elle est restée longtemps l'outil efficace de «design» des fosses. Jusqu'à tout récemment, plusieurs auteurs se sont intéressés à cette méthode : Gauthier et Gray (1971), Marino et Slama (1972), Korobov (1973), Phillips (1973), David et al. (1974), Lemieux (1979), Berlanga et al. (1989), Dowd et Onur (1993) et Wright (1999).

Le second type d'heuristiques est basé sur le concept de la programmation dynamique. Ces méthodes ont pris naissance à la suite de l'algorithme 2-D de Lerchs et Grossmann puis se sont développées afin de traiter le problème tridimensionnel. Ces méthodes sont exactes lorsqu'elles restent dans le domaine du 2-D. Plusieurs tentatives ont été faites pour adapter ces méthodes aux problèmes en tois dimensions; elles ont toutes abouti à l'élaboration d'une méthode heuristique. Bien que plusieurs auteurs se soient penchés sur la question, Lerchs et Grossmann (1965), Johnson et Mickel (1970), Johnson et Sharp (1971), Koenigsberg (1982), Barnes et Johnson (1982), Caccetta et Giannini (1986), Yamatomi et al. (1995) et Frimpong et Achireko (1997), ces méthodes n'ont pas été grandement utilisées comme outil de planification dans l'industrie minière parce qu'elles ont fait leur apparition durant la même décennie que les méthodes exactes en 3-

D. Les heuristiques basées sur une approche de programmation dynamique garantissent de meilleures solutions que celles du cône mobile, mais elles sont généralement plus difficiles à comprendre et à implanter.

En 1965, Lerchs et Grossmann sont les premiers à présenter une méthode optimale pour le problème des contours ultimes. Basé initialement sur la théorie des graphes, l'algorithme de Lerchs et Grossmann fut repris tout d'abord par Johnson (1968) puis par Picard (1976) qui démontra qu'on pouvait modéliser le problème comme celui d'un problème de flot maximal dans un graphe. Plusieurs articles portent sur le sujet, Johnson et Barnes (1988), Plasse et Elbrond (1988), Giannini et al. (1991), Yegulalp et Arias (1992), Jiang (1995) et Hochbaum et Chen (1998).

Une fois les contours ultimes déterminés, quelle que soit la technique utilisée pour y arriver, il devient possible d'entreprendre la résolution du problème de la planification à moyen terme.

3.1.1 Cônes mobiles

La méthode du cône mobile utilisée pour la résolution du problème des contours ultimes d'une mine à ciel ouvert a tout d'abord été proposée par Pana en 1965. Les algorithmes basés sur cette méthode sont les plus répandus dans l'industrie minière à cause de leur simplicité, tant sur le plan de la compréhension que sur le plan de l'implantation informatique.

Une heuristique du cône mobile est un processus d'essais et erreurs. Voici l'idée de base de cette méthode. On choisit d'abord un bloc de valeur positive qui servira de base pour le cône. Le cône représente l'ensemble des blocs qui doivent obligatoirement être extraits pour pouvoir extraire le bloc de base. Les contours générés s'ajustent habituellement aux pentes maximales permises des talus (voir la figure 3.1). Si la valeur du cône est positive, c'est-à-dire que la somme des valeurs des blocs compris à l'intérieur du cône est positive, alors le cône est extrait sinon on cherche un autre bloc de valeur

positive. Si le cône est extrait, on change la topographie de surface de façon à simuler l'extraction du cône. Puis on recommence avec un nouveau bloc de valeur positive. L'algorithme du cône mobile prend fin lorsque tous les cônes à valeur positive ont été évalués

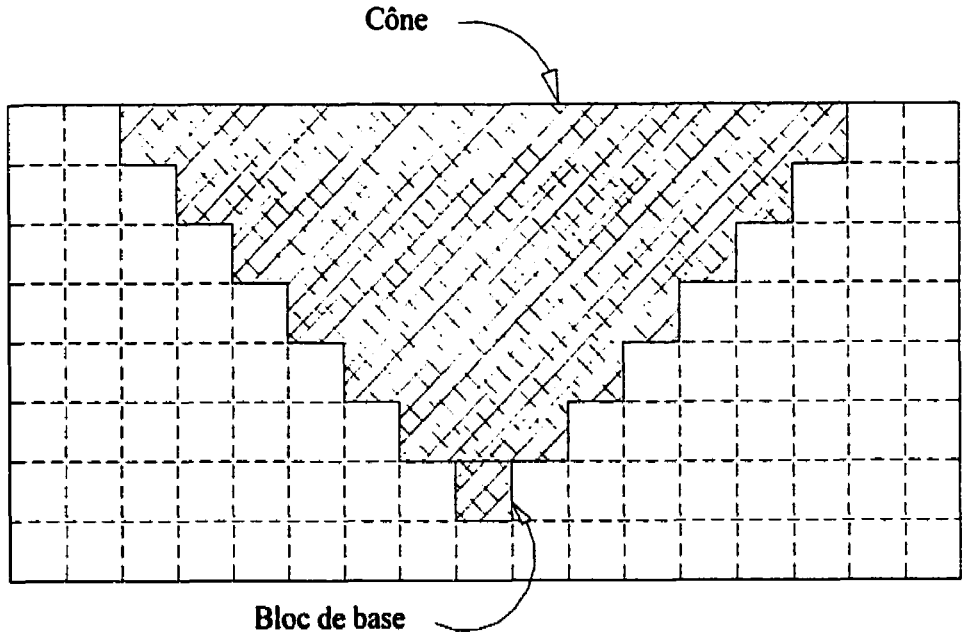


Figure 3.1 : Cône et bloc de base

Malgré sa popularité, cette méthode présente deux inconvénients majeurs. Tout d'abord il est possible que des blocs de base non adjacents puissent être combinés et qu'ensemble ils donnent un cône dont la valeur soit positive. Ainsi la méthode du cône mobile ne tient pas compte de la contribution possible des autres cônes. De plus, il n'est pas possible de regarder toutes les combinaisons de blocs de base adjacents. Gauthier et Gray (1971) rapportent des cas où le cône mobile ne donne pas une solution optimale à cause de ces inconvénients.

En 1973, Phillips présente une variation de la méthode du cône mobile qui tente de pallier à certaines lacunes de la méthode originale. Son algorithme traite le gisement

niveau par niveau et permet de combiner deux ou plusieurs blocs positifs sur un même niveau pour obtenir un cône positif. Toutes les combinaisons possibles de blocs sont regardées. L'inconvénient de cette méthode est que, s'il existe un bloc sur un niveau inférieur qui puisse contribuer à l'enlèvement du cône, ce dernier n'est pas considéré. Lemieux, en 1979, a lui aussi proposé une variation de la méthode du cône mobile qui permet de regarder la combinaison de deux blocs adjacents.

Korobov en 1973 a proposé, quant à lui, une variante de la méthode du cône mobile qui permet de définir plusieurs pentes et non seulement des pentes à 45 degrés. De plus, cette méthode permet de regarder la combinaison de tous les blocs positifs possibles. L'idée est de trouver tous les blocs positifs et de les faire payer pour les blocs négatifs qui se trouvent au-dessus d'eux. La façon de procéder est la suivante : pour chacun des blocs positifs identifié, on soustrait la valeur des blocs négatifs qui se trouvent au-dessus jusqu'à ce que la valeur du bloc positif ou celle de tous les blocs négatifs devienne nulle. Après avoir implanté l'algorithme, on s'est très vite rendu compte que la méthode ne donnait pas toujours la solution optimale. En fait, la solution qu'on obtenait dépendait de l'ordre dans lequel les blocs étaient traités. Toutefois, il s'agissait d'un algorithme extrêmement simple et rapide comparativement à celui de Lerchs et Grossmann, sans pour autant présenter les désavantages de l'algorithme du cône mobile. Dowd et Onur, en 1993, ont proposé une modification à l'algorithme de Korobov qui permet de résoudre cette lacune. Il semblerait, d'après les auteurs, que la méthode, bien que toujours une heuristique, soit comparable à celle de Lerchs et Grossmann en termes de résultats, mais beaucoup plus rapide et simple que cette dernière.

Wright, en 1999, a développé un nouvel algorithme qu'il a nommé «Moving cone II». L'innovation majeure de cet algorithme est qu'il permet l'extraction des cônes négatifs. La méthode est similaire à celle de Korobov, mais l'approche est différente. Ce qu'il propose, c'est de dessiner une courbe de la valeur de la fosse chaque fois qu'un cône est extrait. Cette courbe, à un certain moment, atteint une valeur positive optimale puis

descend constamment sans jamais remonter par la suite. Une fois que tous les blocs positifs ont été extraits, on garde la dernière valeur positive de la fosse sur la courbe.

Les heuristiques du cône mobile ont été très utilisées durant les années 60 et jusqu'au milieu des années 80.

3.1.2 Lerchs et Grossmann 2-D et la programmation dynamique

Dans l'article paru en 1965, Lerchs et Grossmann présentent deux méthodes de résolution des contours ultimes. La première des deux méthodes se présente sous la forme d'un algorithme simple de programmation dynamique pour une fosse en deux dimensions (ou pour une section verticale de la fosse) et la deuxième méthode présente un algorithme plus élaboré pour les problèmes plus généraux à trois dimensions. Les étapes de l'algorithme de Lerchs et Grossmann 2D sont présentées en détail en Annexe I.

La programmation dynamique est une approche qui transforme un problème complexe en une séquence de problèmes plus simples. La caractéristique principale de cette approche est sa procédure à niveaux et états multiples. Les méthodes de résolution de contours ultimes basées sur la programmation dynamique utilisent à la base l'algorithme de Lerchs et Grossmann 2-D qui est optimal. Théoriquement, par programmation dynamique, nous serions en mesure de résoudre ce problème et d'obtenir des solutions optimales. En pratique pourtant, le nombre de combinaisons de blocs possibles rend l'application en trois dimensions de ces types d'algorithmes pratiquement impossible.

Beaucoup d'auteurs ont tenté d'adapter Lerchs et Grossmann 2D au problème en trois dimensions; les méthodes ainsi développées sont classées comme des heuristiques parce qu'elles ne reflètent pas la réalité en trois dimensions. Johnson et Mickel (1970), Johnson et Sharp (1971), Barnes et Johnson (1982) et Braticevic (1984) en sont de bons exemples.

Koenigsberg, en 1982, a été le premier à développer un algorithme basé sur la programmation dynamique pour résoudre le problème réellement en trois dimensions

bien qu'il se soit inspiré à la base de l'algorithme de Lerchs et Grossmann 2D. Toutefois, cet algorithme présentait une lacune majeure qui pouvait produire des solutions dégénérées. Un peu plus tard, Wilke et Wright (1984) proposent une solution au problème de dégénérescence, mais les temps de résolution ne sont pas acceptables. Par la suite, Wright (1987) propose des changements à l'algorithme de 1984 pour réduire les temps de résolution, mais ces modifications font en sorte que l'algorithme n'est désormais plus optimal. Yamatomi et al. en 1995 reprennent l'article de Wilke et Wright (1984) et y apportent quelques modifications qui en font une heuristique.

Barnes et Johnson (1982) ainsi que Caccetta et Giannini (1986) ont proposé une approche différente. Plutôt que d'essayer de déterminer les contours ultimes, ils ont proposé de borner le problème afin de réduire le temps de calcul. Il s'agit en fait d'éliminer les blocs qui ne pourront jamais faire partie de la fosse et d'extraire immédiatement ceux qui en feront partie de toute façon. Par la suite on applique n'importe lequel des algorithmes qui nous permet de trouver les contours ultimes sur les blocs restants. D'après Barnes et Johnson (1982), la méthode de programmation dynamique développée par Koenigsberg (1982) serait la plus efficace pour trouver de bonnes bornes. Ils proposent toutefois six autres méthodes, dont la majorité font appel à Lerchs et Grossmann 2D, qui peuvent aussi servir pour trouver les bornes. Caccetta et Giannini (1986), quant à eux, utilisent l'algorithme développé par Johnson et Sharp (1971).

3.1.3 Lerchs et Grossmann 3-D et la théorie des graphes

Lerchs et Grossmann ont développé la première méthode optimale en trois dimensions pour déterminer les contours ultimes d'une mine à ciel ouvert. Il s'agit de la deuxième partie de leur article de 1965. L'algorithme de Lerchs et Grossmann 3D est séparé en deux parties. La première partie sert à construire le graphe et la deuxième partie sert à calculer la fermeture de poids maximal sur le graphe.

Lors de la première partie, on construit le graphe initial en traçant des arcs orientés entre chacun des nœuds et les 9 nœuds qui se trouvent au-dessus de lui. Supposons, par exemple, que nous avons le gisement en deux dimensions présenté à la figure 3.2 :

-2	-2	-2	-2	-2	3	-2	-2
-2	-2	8	-2	-2	-2	-2	-2
-2	-2	6	-2	5	-2	-2	-2

Figure 3.2 : Exemple de gisement

Chacun des blocs de la figure précédente devient un nœud du graphe. À chaque nœud du graphe est associée une valeur représentée par le coût ou le profit généré par l'extraction du bloc correspondant. On peut associer à ce gisement le graphe des préséances présenté à la figure 3.3 :

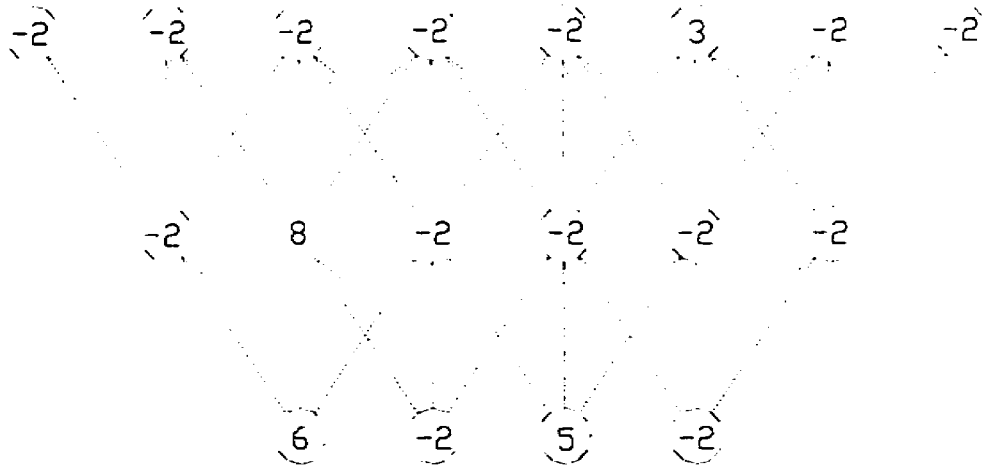


Figure 3.3 : Graphe représentant les contraintes de préséance entre les blocs

Pour terminer, on doit relier tous les nœuds à un nœud artificiel nommé *s*. On forme ainsi un arbre où le nœud artificiel *s* est la racine de l'arbre. La figure 3.4 montre l'arbre initial ainsi créé.

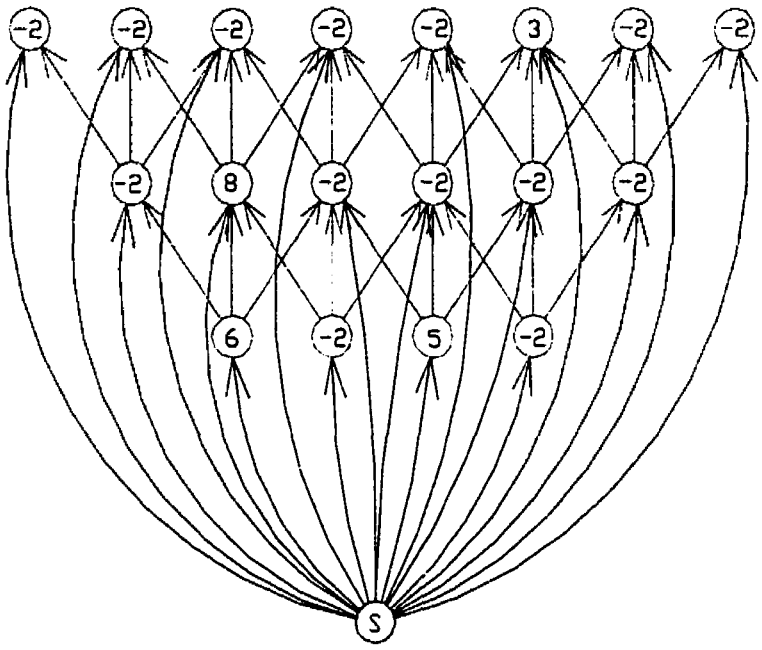


Figure 3.4 : Arbre initial du graphe de Lerchs et Grossmann

La deuxième partie de l'algorithme consiste à trouver la fermeture maximale sur ce graphe. Le détail complet de l'algorithme de Lerchs et Grossmann est donné en Annexe II.

Même si l'algorithme de Lerchs et Grossmann a fait ses preuves depuis maintenant 35 ans et qu'il s'agit de l'algorithme de référence à partir duquel tous les nouveaux algorithmes développés sont classés et jugés, il n'était presque pas utilisé dans l'industrie avant le milieu des années 80. Les raisons majeures qui expliquent ce phénomène, étaient de nature pratique. Comme l'algorithme de Lerchs et Grossmann est très complexe, il est difficile à programmer et difficile à comprendre. On ne pouvait pas non

plus y introduire de contraintes comme les pentes ou les routes de halage et les temps de résolution étaient très élevés.

Au cours des années, toutefois, il a été remodelé et modifié fréquemment afin de le rendre plus efficace. Stuart (1992), Zhao et Kim (1992) et le logiciel Whittle en sont de bons exemples. Ils ont tous, à un certain niveau, réussi à outrepasser certaines limitations mentionnées plus haut comme d'incorporer des pentes variables lors des calculs et réduire les temps de résolution.

Avec l'avènement du logiciel commercial Whittle, la méthode de Lerchs et Grossmann est devenue de plus en plus populaire dans l'industrie et ce, depuis le milieu des années 80 jusqu'à maintenant, soit 20 ans après la parution de l'article original en 1965.

3.1.3.1 Les pentes variables

Pendant longtemps, les différentes méthodes développées au cours des ans pour déterminer les contours ultimes n'ont pas été favorisées dans l'industrie à cause des problèmes d'implantation qu'elles présentaient, comme leur incapacité à prendre en considération des pentes de talus variables pour différentes parties de la fosse.

Il semblerait, selon Lipekewich et Borgman (1969), qu'un certain Charles Johnson, un étudiant post gradué de l'université de Californie en 1967, fut le premier à observer qu'un patron fixe de cinq ou neuf prédécesseurs ne donnait pas exactement les résultats escomptés en termes de pentes. Afin d'obtenir des pentes de 45°, ce dernier avait proposé un patron qui incluait les blocs se trouvant à l'intérieur d'un mouvement de cavalier aux échecs. Ainsi, dans le graphe des prédécesseurs, il n'y avait plus cinq ou neuf arcs pour un bloc mais treize. Bien que cette façon de procéder ne soit valable que pour des pentes fixes à 45°, l'idée de base de la recherche des prédécesseurs par patron conique était lancée. Lipekewich et Borgman, quant à eux, ont repris l'idée et ont modifié l'algorithme de Lerchs et Grossmann afin d'inclure le nouveau patron aux

calculs. Il semblerait, selon leurs résultats, que le fait d'inclure un patron de recherche à l'intérieur de leurs calculs diminue l'espace mémoire nécessaire.

En 1977, Chen reprend l'article de Lipekewich et Borgman (1969) et généralise le concept pour inclure dans les calculs de Lerchs et Grossmann des pentes variables. Pour y arriver, il se sert d'une paire de données qui représentent l'azimut et le pendage de la pente désirée. Il peut y avoir autant de paires de données qu'il y a de pentes différentes voulues. Les avantages de cette façon de procéder sont que la solution trouvée demeure optimale et que la valeur des pentes est indépendante des dimensions des blocs.

En 1988, Caccetta et Giannini écrivent un article qui décrit le mieux l'algorithme de recherche minimum des prédécesseurs. Ils y expliquent très bien ce qui a été fait par les 2 premiers auteurs et proposent un troisième algorithme qui inclut des paramètres de tolérance pour les pentes et qui s'applique aussi bien à l'algorithme de Lerchs et Grossmann qu'à l'algorithme du flot maximum.

3.1.4 Flot maximum

En 1965, lorsque Lerchs et Grossmann ont proposé leur algorithme, ils ont spécifié que le problème pouvait être traité de plusieurs façons différentes, soit à l'aide de la programmation dynamique, soit à l'aide de la théorie des graphes (fermeture maximale d'un graphe orienté), soit comme un problème de flot ou encore comme une analogie hydrostatique (problème de gestion de projet).

En 1968, dans le cadre de sa thèse de doctorat, Johnson fut le premier à reconnaître la relation entre le problème des contours ultimes et le flot maximum. Il voulait en fait déterminer des séquences de production. Pour y arriver, il avait réduit le problème des contours ultimes présenté par Lerchs et Grossmann en un autre problème de fermeture, mais cette fois sur un graphe biparti. Le graphe biparti était construit de la façon suivante : un arc était créé si et seulement s'il existait un chemin direct entre un nœud de poids positif et un nœud de poids négatif. Selon Hochbaum et Chen (1998), il avait en

fait décrit le problème des contours ultimes comme un problème de transport. Puis il observa que ce problème pouvait être résolu comme un problème de flot maximum.

En 1976, Picard a démontré mathématiquement que la méthode présentée par Lerchs et Grossmann, qui consistait à trouver la fermeture maximale sur un graphe, était équivalente à trouver un flot maximum sur un graphe adapté au problème des contours ultimes. Le réseau décrit par Picard est construit à partir du graphe de Lerchs et Grossmann. Le nouveau graphe est obtenu en ajoutant un nœud source *s* et un nœud puits *t*. Le nœud source est relié à tous les nœuds *i* qui ont une valeur positive par un arc orienté *(s, i)* de capacité égale à la valeur du nœud. Le nœud puits est relié à tous les nœuds *j* qui ont une valeur négative ou nulle par un arc orienté *(j, t)* de capacité égale à la valeur absolue du nœud. Dans ce nouveau graphe, les arcs de préséance ont une capacité infinie. La figure 3.5 présente le graphe de Picard construit à partir du gisement présenté dans la figure 3.2 dans la section qui porte sur Lerchs et Grossmann 3D. Comme on peut le voir à la figure suivante, tous les arcs du graphe des prédécesseurs portent une capacité infinie, alors que tous les arcs qui entrent au nœud puits ont une capacité de 2 (soit la valeur absolue des blocs de stérile dans cet exemple) et les arcs qui sortent de la source ont une capacité égale à la valeur des blocs de minerai.

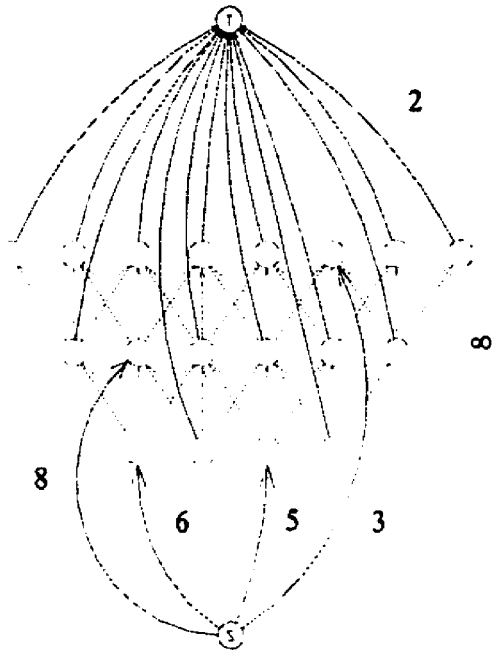


Figure 3.5 : Graphe pour le calcul des contours ultimes à l'aide d'un flot maximum

Picard a ensuite montré l'équivalence de sa méthode avec celle de ses prédécesseurs. En fait, la fermeture maximale recherchée sur le graphe de préséance de Lerchs et Grossmann correspond à l'ensemble des nœuds du côté de la source dans la coupe de capacité minimale déterminée par un algorithme de flot maximum sur le nouveau graphe proposé par Picard.

L'efficacité de la méthode proposée par Picard repose sur le choix adéquat de l'algorithme de flot maximum qui sera utilisé lors de la résolution du problème. Tachefine (1993) et Hochbaum et Chen (1998) ont fait des tests d'efficacité afin de déterminer l'algorithme de flot maximum qui est le plus efficace pour résoudre ce type de problème. D'après les résultats de Hochbaum et Chen, ce serait la catégorie des algorithmes de «push-relabel» qui présenterait les algorithmes les plus prometteurs. Tachefine, lui, va plus loin en retenant le «*Highest Label Preflow Algorithm*» comme étant l'algorithme le plus efficace pour la résolution du problème de flot maximum sur les graphes représentant des gisements miniers.

La méthode qui utilise le flot maximum pour déterminer les contours ultimes a vraiment connu son essor lors des dix dernières années. Comme les développements se sont faits tardivement (voir Johnson et Barnes 1988, Plasse et Elbrond 1988, Giannini et al. 1991, Yegulalp et Arias 1992, Jiang 1995 et Tachefine 1997), elle n'a pratiquement jamais été utilisée dans l'industrie. Selon Jiang (1995), ce qui expliquerait ce phénomène, c'est qu'au moment de la parution des articles de Johnson et de Picard, le seul algorithme de flot maximum disponible était celui de Ford et Fulkerson. Bien que cet algorithme se termine après un nombre fini d'itérations, le temps de résolution n'en demeurait pas moins très élevé. Or, depuis une dizaine d'années, des algorithmes de flot maximum de plus en plus puissants ont été développés. De plus, la méthode de flot maximum, lorsqu'elle a été proposée la première fois, demandait plus de mémoire vive que celle de Lerchs-Grossmann et, tout comme avec cette dernière, il était impossible de tenir compte de pentes variables lors du «design» des fosses. Mais, comme nous l'avons vu précédemment, ce n'est plus un problème si on établit le graphe des préséances à l'aide de la méthode de recherche des prédécesseurs par patron conique (Chen 1977).

Il semblerait que l'algorithme de flot maximum combiné à la technique pour déterminer les bornes proposée par Barnes et Johnson (1982) soit beaucoup plus simple à comprendre et à programmer que celui de Lerchs et Grossmann et aussi rapide, sinon plus, que la méthode des cônes mobiles (Jiang 1995, Giannini et al. 1991).

3.1.5 Les autres méthodes

Quelques autres méthodes très peu populaires et presque pas étudiées ont été proposées pour résoudre le problème des contours ultimes.

Tout d'abord, il y a eu la formulation comme un problème de transport. Comme nous l'avons vu dans la section précédente, c'est Johnson (1968) qui a été le premier à formuler le problème sous cette forme puis plus récemment, Huttagosol et Cameron (1992) ont repris l'idée. Les détails concernant la façon de construire le graphe ont été formulés dans la section consacrée au flot maximum. L'objectif du problème de

transport est d'envoyer des quantités à partir des fournisseurs (sources) vers des consommateurs (destinations) de façon à ce que toutes les demandes soient satisfaites sans excéder l'offre et ce, en minimisant les coûts de transport. Huttagosol et Cameron ont utilisé la méthode du simplexe réseau (normalement utilisée pour les problèmes de flot à coût minimum, qui sont une généralisation des problèmes de transport) pour résoudre leur problème et les résultats montrent que cette façon de procéder est plus lente que l'algorithme de Lerchs et Grossmann.

Meyer, en 1969, a formulé le problème des contours ultimes comme un problème de programmation linéaire. Théoriquement, cette façon de formuler le problème nous mène à l'optimal. Toutefois, en pratique, il est impossible de traiter des problèmes de très grande taille avec ce modèle en des temps raisonnables.

En 1993, Schofield et Denby ont proposé d'utiliser la méthode heuristique des algorithmes génétiques pour résoudre le problème des contours ultimes. Les algorithmes génétiques imitent les opérations génétiques et le phénomène de sélection naturelle dans leur recherche d'une solution optimale. Il s'agit d'un développement relativement récent dans le domaine de l'intelligence artificielle et qui a commencé à être utilisé depuis la fin des années 80. L'algorithme débute avec une population de solutions aléatoires et tente de faire évoluer cette population en créant une série de générations à l'aide des techniques de probabilités et d'opérateurs génétiques. L'algorithme est basé sur les étapes suivantes: les chromosomes codent la structure des organismes, l'évolution opère sur les chromosomes, la sélection naturelle s'assure que les bons chromosomes se reproduisent plus que les mauvais, la reproduction est la clef de l'évolution et finalement, les mutations et les recombinaisons font que les chromosomes enfants sont différents des chromosomes parents. L'objectif du système est de produire des fosses et des cellules d'extraction qui optimisent la valeur présente nette, d'éviter toutes les hypothèses qui peuvent mener à un raisonnement circulaire, d'être efficace en terme de temps de calcul et d'être flexible.

Finalement, en 1997, Frimpong et Archireko ont proposé de résoudre le problème des contours ultimes à l'aide de réseaux de neurones. Il semblerait que les résultats soient les même que ceux qu'on obtiendrait à l'aide de l'algorithme de LG 2D.

3.2 Séquences d'exploitation

Selon Sevim et Lei (1995), la planification dans une mine à ciel ouvert requiert la détermination de quatre variables majeures soit: 1) la teneur de coupure, 2) le taux de production, 3) la séquence de minage ou d'exploitation et 4) les contours ultimes de la fosse. La durée de vie de la mine peut être considérée comme une cinquième variable. Mais une fois que les quatre variables mentionnées ci-haut sont connues, on peut la déduire automatiquement. Les quatre principales variables interagissent de façon circulaire entre elles, c'est-à-dire que la teneur de coupure doit être connue pour déterminer la nature des blocs et ainsi trouver les contours ultimes. Les contours ultimes doivent être évalués pour produire une séquence d'exploitation qui comprend les taux de production et la séquence de minage. Finalement, on utilise les taux de production et la séquence de minage pour calculer les revenus (en valeur présente nette) et les coûts de production qui, eux, vont à leur tour permettre de déterminer la teneur de coupure économique. Ainsi, on ne peut pas déterminer l'une de ces variables sans que la variable précédente ait préalablement été déterminée. Idéalement, l'optimisation devrait se faire de façon itérative pour fixer chacune des variables jusqu'à ce que l'optimum soit atteint. Mais comme nous l'avons vu précédemment, la résolution du problème des contours ultimes dans un temps raisonnable est un exploit en soit, et comme nous le verrons dans les sections suivantes, la détermination de la meilleure séquence de minage est tout aussi difficile. De plus, s'ajoutent des contraintes liées à la production, comme la planification des routes de halage, des bancs de travail, du déplacement des pelles, etc. Ainsi, la solution produite par la méthode d'optimisation doit être non seulement optimale mais aussi opérationnelle.

Il est donc évident qu'il n'est pas facile, sinon présentement impossible, d'obtenir une telle solution. C'est pourquoi le problème est généralement solutionné en posant des hypothèses et en fixant une ou plusieurs variables. La plupart des méthodes qui ont été développées jusqu'à présent vont suivre l'un des scénarios suivants:

- Les contours ultimes sont déterminés en considérant une teneur de coupure ainsi qu'un taux de production fixés *a priori*. Puis une séquence de minage est trouvée de façon plutôt instinctive à l'aide de méthodes manuelles d'essais et erreurs ou encore à l'aide de logiciels maison qui utilisent souvent des techniques de programmation linéaire permettant de satisfaire à la production à court terme de la mine.

- Une teneur de coupure ainsi qu'un taux de production sont fixés et les contours ultimes sont déterminés. Puis, finalement, une séquence d'extraction est définie à l'aide d'une heuristique spécialement conçue à cet effet.

- Les contours ultimes sont déterminés en fonction d'un certain prix de métal maximum et d'une certaine teneur de coupure maximale. Puis une série de contours ultimes sont calculés en faisant varier l'un ou l'autre des paramètres ou les deux. On obtient ainsi une série de fosses emboîtées l'une dans l'autre qui permettent de déterminer une séquence de production pour un taux de production fixé.

- Une teneur de coupure et un taux de production sont fixés, puis les contours ultimes et les séquences de minages sont déterminés de façon itérative.

Il aura fallu du temps avant de s'attaquer au problème des séquences de production. C'est seulement vers le début des années 80 que le problème suscite de l'intérêt auprès du domaine de la recherche opérationnelle. Et encore là, très peu de personnes osent aborder le problème. C'est au cours des années 90 que l'intérêt pour ce problème prend vraiment de l'ampleur. Lorsqu'on se rend compte de la nature circulaire du problème de planification à long terme, on abandonne un peu la résolution des contours pour se consacrer un peu plus à celui des séquences de production. Présentement il existe,

comme nous le verrons dans les sections qui suivent, quelques bonnes méthodes qui peuvent résoudre ce problème. Toutefois, il est très difficile dans ce domaine de parler de méthodes optimales ou heuristiques. Si on se réfère à la description du problème qui a été faite précédemment, aucune solution présentement disponible n'est optimale. Une division des algorithmes a quand même été faite selon le type de méthodes et le scénario qu'elles utilisent.

3.2.1 Les méthodes heuristiques et d'essais et erreurs

Les méthodes les plus répandues pour déterminer les séquences d'exploitation sont des heuristiques. Les méthodes heuristiques procèdent généralement par essais et erreurs et les calculs sont souvent faits à la main ou à l'aide d'un logiciel interactif développé par la mine qui aide à faire les calculs. Voici les étapes qui caractérisent les méthodes heuristiques:

1) On détermine tout d'abord les contours ultimes de la fosse à l'aide d'une des méthodes présentées à la section 3.1.

2) À l'aide de l'information fournie par les cartes topographiques, des teneurs de coupure au concentrateur et des contraintes reliées à l'extraction du minerai, on détermine quelques séquences de production pour chacune des années et on choisit la meilleure ou celle qui présente la valeur présente nette la plus élevée parmi celles qui ont été développées.

On retrouve dans la littérature quelques auteurs qui proposent de résoudre le problème des séquences de production de cette façon : Leigh et Blake (1970), Gershon (1987), Zhang et Yang (1990) et Bohnet (1990). Robinson et Prenn (1972), Wilke, Mueller et Wright (1984), Fytas, Calder et Pelley (1987) et Fytas, Hadjigeorgious et Collins (1994) sont tous des auteurs qui ont utilisé le processus de simulation pour générer un plus grand nombre de séquences de production. Ces heuristiques ont de grands inconvénients :

- Les séquences de production obtenues à l'aide des méthodes heuristiques et qui sont basées sur les essais et erreurs tendent à être arbitraires, c'est-à-dire que pour une fosse donnée, plusieurs séquences de production réalisables peuvent être générées. Obtenir une bonne séquence dépend avant tout de l'habileté et de l'expérience du planificateur.

- On établit tout d'abord une séquence d'exploitation du gisement sans tenir compte de la valeur de l'argent dans le temps. Puis on calcule la valeur présente nette du gisement en fonction de la séquence d'exploitation qui a été établie. On peut ainsi faire plusieurs scénarios et ensuite choisir le plus avantageux. Or, les contours ultimes de la fosse sont étroitement liés à la séquence de minage.

- Même si on fait plusieurs scénarios de minage et qu'on choisit celui qui donne la plus grande valeur présente nette parmi les quelques-unes qui ont été générées, il n'y a aucune garantie que la meilleure solution possible pour l'exploitation en question se trouve parmi celles qui ont été analysées.

Seule, à mon avis, la méthode proposée par Gershon (1987) présente un certain intérêt. Sa méthode heuristique se base un peu sur le principe de la méthode des cônes mobiles pour déterminer dans quel ordre les blocs seront extraits. En fait, il a appelé cette méthode «ranked positional weight». Tout d'abord, on commence par déterminer des contours ultimes. Puis à partir de la topographie actuelle, on trouve tous les blocs qu'il est possible d'extraire et on leur attribue un poids. Le poids d'un bloc est calculé en générant un cône inversé qui part du bloc considéré et qui inclut tous les blocs qui se trouvent en dessous de lui jusqu'aux limites de la fosse. La valeur résultant de l'addition de tous les blocs qui font partie du cône donne une mesure de l'avantage d'extraire ce bloc à ce moment-là de la production. Ensuite, tous les blocs sont placés en ordre décroissant et un choix du bloc à extraire est fait parmi ceux qui ont une valeur plus élevée. On extrait le bloc choisi selon certains critères, on redéfinit une nouvelle topographie et on recommence cette procédure jusqu'à ce que tous les blocs de la fosse

aient été extraits. En fait, on extrait le bloc qui nous permettra d'atteindre le profit le plus rapidement possible. Les critères de choix de bloc dépendent de contraintes liées à la production, comme éviter d'extraire un bloc trop éloigné du bloc choisi à l'itération précédente.

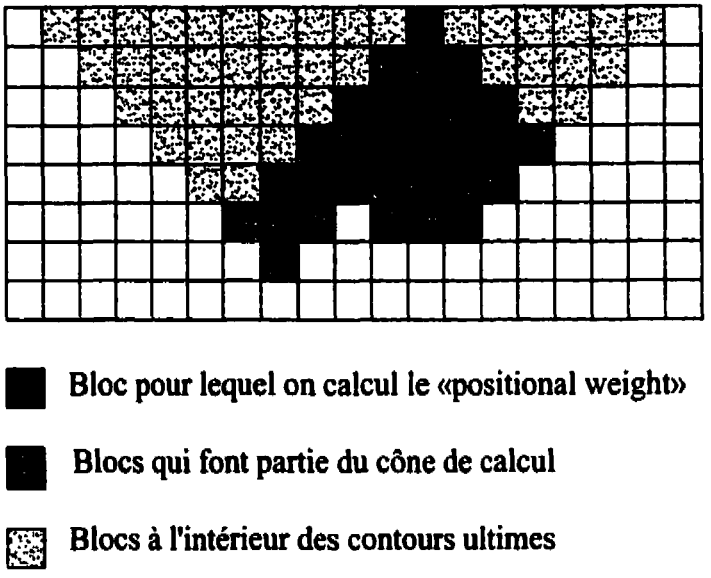


Figure 3.6 : Séquence d'extraction selon la méthode heuristique de Gershon

3.2.2 La programmation linéaire

La programmation linéaire a été utilisée de trois façons différentes pour résoudre le problème des séquences de production. La première méthode se divise en deux phases : 1) on utilise une heuristique où on détermine les contours ultimes puis, 2) on détermine une séquence de minage à l'aide d'un logiciel maison ou commercial qui utilise des techniques de programmation linéaire permettant de satisfaire à la production à court terme de la mine (voir Gershon (1987) et Blackwell (1993)). Cette façon de procéder est surtout utilisée lorsqu'il y a des contraintes de mélange à respecter, c'est-à-dire lorsqu'il faut tenir compte des caractéristiques du minerai.

La deuxième façon de procéder a été proposée par Gershon en 1982. Il a formulé le problème totalement comme un problème de programmation linéaire en nombres entiers. Le problème des séquences de production peut être formulé de cette façon, bien qu'on ne puisse obtenir des résultats que pour des problèmes de petite taille. Gershon a tenté de contourner cette limitation en réduisant le nombre de blocs. Pour ce faire, il a utilisé des logiciels commerciaux qui pouvaient traiter des problèmes de l'ordre de 8 190 contraintes et de 131 000 variables. Pour résoudre le problème, il a donc dû le fractionner en plusieurs parties qui permettent chacune d'obtenir la production à long, à moyen et à court terme. Pour les séquences à long terme, il utilise de très gros blocs qu'il regroupe en périodes. Par exemple, un projet de 20 ans peut être divisé en tranches de 5 ans. Puis, pour une des tranches de 5 ans, il utilise des blocs plus petits et il optimise pour chaque année. Enfin, une autre division est faite pour des périodes de 3 mois et finalement pour des périodes hebdomadaires. Il y a de grands inconvénients à utiliser une méthode de ce type. Premièrement, l'optimisation doit être faite de façon régulière et, deuxièmement, on ne tient nullement compte de la nature circulaire du problème. Toutefois, il est possible d'introduire différentes contraintes de production à l'intérieur du modèle, ce qui permet d'obtenir des solutions physiquement réalisables.

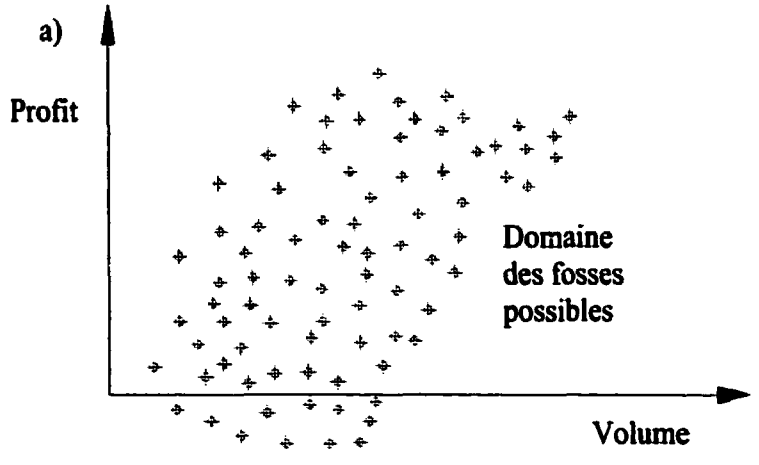
La troisième façon de déterminer des séquences de production à l'aide de la programmation linéaire a été proposée par Johnson en1968. Il a utilisé la programmation linéaire comme sous-problème dans un algorithme de décomposition qui permet de traiter le problème à plusieurs niveaux (multi-périodes) comme nous verrons dans une section subséquente.

3.2.3 L'analyse paramétrique

Il existe trois types d'analyse paramétrique: la paramétrisation, la paramétrisation des réserves et finalement la paramétrisation Lagrangienne que nous verrons plus en détail dans la section des méthodes de décomposition. En général, on qualifie d'analyse paramétrique les méthodes qui font varier un ou plusieurs paramètres lors du processus

d'optimisation. Dans ces méthodes, on fait généralement varier la valeur des paramètres suivants : le prix du minerai, la teneur de coupure, le taux de production, etc. Certains le font de façon purement heuristique tandis que d'autres clament le faire de façon optimale. Quoi qu'il en soit, le principe reste le même et on pourrait toutes les classer comme des méthodes heuristiques.

Vallet (1976) et Coléou (1988) illustrent très bien le principe de l'analyse paramétrique. En faisant varier les paramètres de définition des fosses, ce qu'on cherche à obtenir c'est un ensemble de fosses possibles (ou réalisables). Si on traçait le domaine de toutes les fosses possibles, on obtiendrait un nuage de points qui ressemble sensiblement à celui de la figure 3.7 (a). Pour un seul contour ultime d'une fosse, il existe une multitude de projets d'exploitation possibles. Chaque projet est caractérisé par sa courbe de profit, figure 3.7 (b). L'analyse paramétrique permet de déterminer la courbe de profit maximal pour les paramètres observés, figure 3.7 (c).



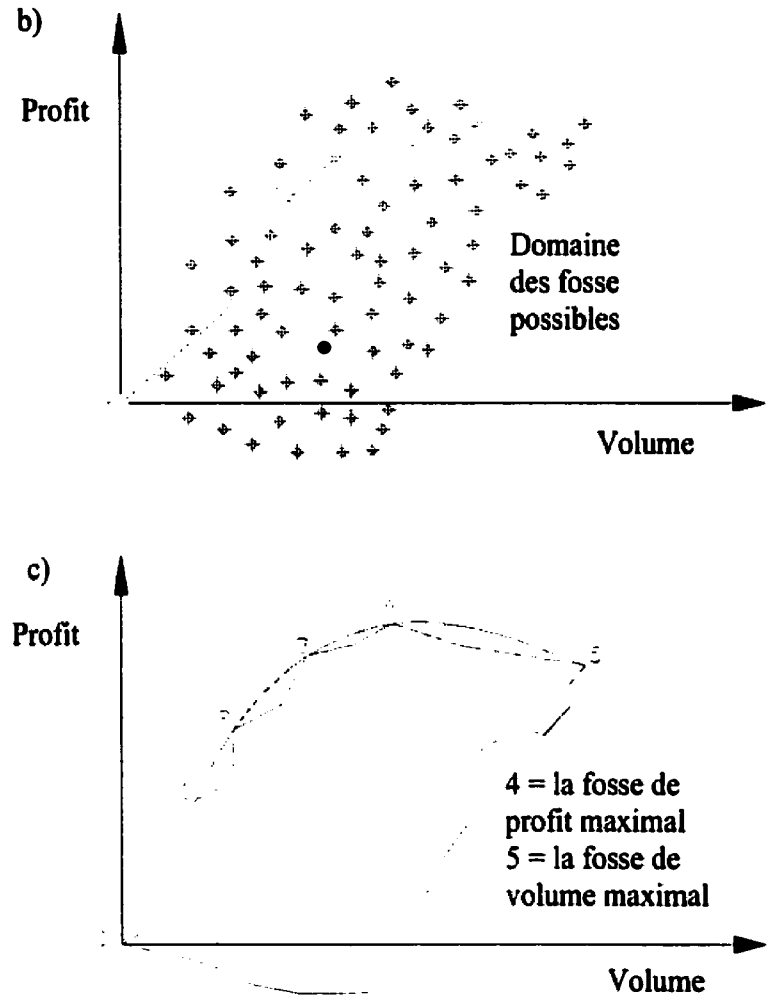


Figure 3.7 : Principe de l'analyse paramétrique. a) Domaine de toutes les fosses possibles, b) Courbe de profit d'un projet pour un contour ultime donné, c) Courbe de profit maximal pour les paramètres observés.

Dans cet exemple, les paramètres considérés sont le volume et le profit. Il s'agit là d'un bon exemple de paramétrisation. En fait, la paramétrisation est basée sur le principe de dominance, c'est-à-dire qu'une solution en domine une autre, ou qu'elle est meilleure qu'une autre si elle procure un meilleur profit et un volume plus petit. D'où la courbe en rouge sur le graphique c). La paramétrisation des réserves, comme nous le verrons un peu plus loin, implique d'autres paramètres.

Hochbaum et Chen (1998) donnent un très bon résumé des méthodes d'analyse paramétrique.

3.2.3.1 La paramétrisation

La première utilisation de l'analyse paramétrique a été proposée par Lerchs et Grossmann en 1965. Ceux-ci ont reconnu que d'avoir un contour optimal pour une fosse n'était pas très pratique s'il n'existait pas une bonne séquence d'extraction pour y parvenir. C'est pourquoi ils ont introduit le concept d'analyse paramétrique qui consiste à faire varier le prix du minerai et à résoudre à chaque fois l'algorithme des contours ultimes de façon à obtenir une série de fosses incluses l'une dans l'autre, appelées couramment «fosses imbriquées». Le principe est simple. Plus le prix de vente du métal est élevé, plus la fosse optimale sera grande et plus le prix est bas, plus la fosse optimale sera petite. En agissant de la sorte, on veut autant que possible extraire le minerai le plus riche au début de l'exploitation de façon à maximiser la valeur présente nette du gisement.

Ainsi on voudra former une série de *n* fosses imbriquées :

$F_1 \subseteq F_2 \subseteq F_3 \subseteq \ldots F_n$ tel que $P_1/V_1 > P_2/V_2 > P_3/V_3 > \ldots P_n/V_n$

où P_i = le profit total de tous les blocs compris dans la fosse F_i

V_i = le volume total de la fosse F_i

En d'autres termes, la fosse F_1 a le plus grand ratio profit/volume et la fosse F_n le plus petit ratio profit/volume. Selon Thomas (1996), la qualité des fosses imbriquées obtenues dépendra grandement, d'une part, du fait qu'il n'y a pas trop de dépendance entre les régions de minerai et de stérile dans le gisement et, d'autre part, de la variation de la valeur des blocs à chaque itération. Avec les fosses trouvées, il est ensuite possible de déterminer une séquence d'exploitation. Rychkun et Chen (1979), Cai et Banfield

(1993) ainsi que Whittle sont tous des auteurs qui ont utilisé de près ou de loin cette méthode.

Le logiciel commercial Whittle, entre autres, utilise cette façon de procéder. Il s'agit en fait d'une compagnie spécialisée exclusivement dans les logiciels de planification minière. Elle a été créée dans le milieu des années 80 par John Whittle. La compagnie offre un logiciel commercial qui utilise l'algorithme de Lerchs et Grossmann pour faire la planification à long terme dans les mines à ciel ouvert. Le logiciel peut calculer à l'aide de l'algorithme de Lerchs et Grossmann quarante à cinquante fosses qui correspondent chacune à un contour optimal pour un prix donné. La figure 3.8 présente un exemple des fosses qui sont calculées.

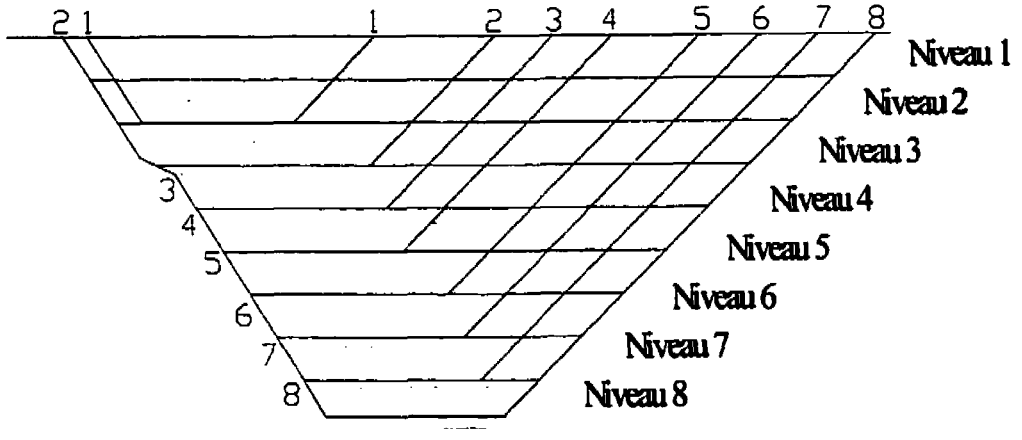


Figure 3.8 : Séquences d'exploitation calculées avec Whittle

Les numéros 1 à 8 représentent les fosses qui peuvent être calculées à l'aide de du logiciel Whittle selon différents prix du métal. Les intersections entre les contours des fosses et le niveau des bancs définissent des sections qui seront extraites. Une fois que nous avons ces sections, on peut établir la séquence de minage selon deux scénarios différents : le pire cas et le meilleur cas.

Le pire cas serait de miner toutes les sections du premier banc en commençant par celui de la plus petite fosse, puis de miner les sections du deuxième banc et ainsi de suite. La figure 3.9 représente ce scénario. Il s'agit du pire cas possible parce qu'on ne va pas chercher le minerai en premier. Les revenus commencent à être considérables seulement vers la fin de la vie de la mine.

Le meilleur cas serait de miner tous les bancs successifs de la plus petite fosse puis de miner les bancs successifs de la deuxième fosse et ainsi de suite. La figure 3.10 représente ce scénario. Il s'agit du meilleur cas possible parce qu'on va chercher le minerai le plus vite possible.

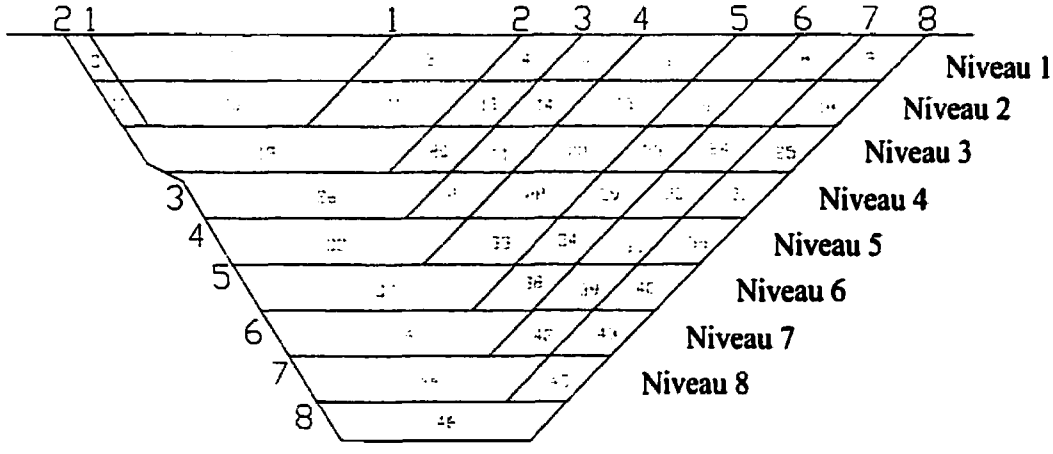


Figure 3.9 : Séquence d'exploitation selon le scénario du pire cas

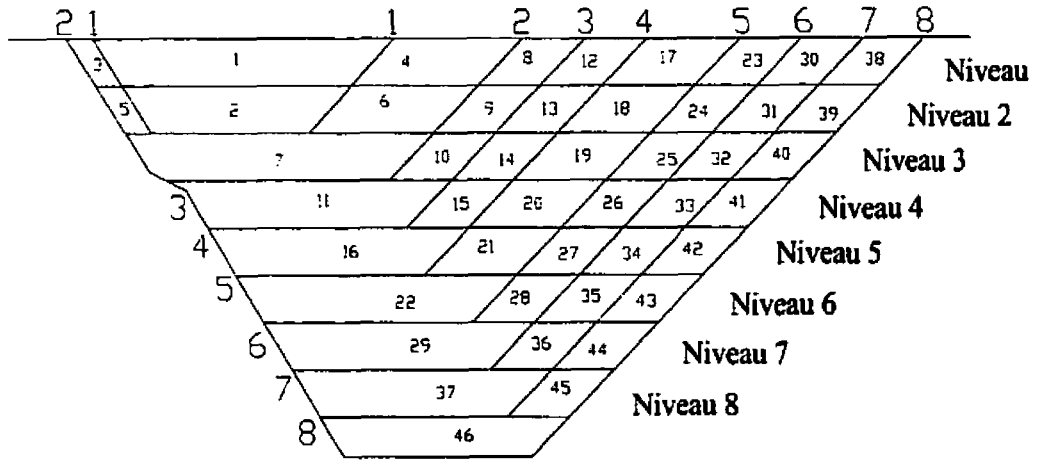


Figure 3.10 : Séquence d'exploitation selon le scénario du meilleur cas

Une fois qu'une séquence d'exploitation est déterminée selon les deux cas mentionnés, il est possible de calculer la valeur présente nette de l'exploitation. Donc, à l'aide de l'outil fourni par le logiciel Whittle, il est possible d'orienter la production vers les endroits qui semblent les plus prometteurs et on sait dès le départ que la valeur présente nette du projet se situera quelque part entre le pire cas et le meilleur cas. Bien que rapide, il ne s'agit pas de la meilleure façon d'obtenir des séquences de production.

Vallet (1976) et Korobov (1993) ont développé des méthodes différentes pour inclure la paramétrisation lors de l'optimisation : toutes les fosses imbriquées sont calculées durant une seule et même itération. Bien que Korobov n'ait pas cité les travaux de Vallet, les deux auteurs font sensiblement la même chose, c'est-à-dire qu'ils ont modifié l'algorithme de Lerchs-Grossmann pour inclure la paramétrisation. Ils y arrivent en cherchant, à chaque étape de l'algorithme, la fosse qui présente le plus grand ratio profit/volume parmi toutes les fosses possibles dans le graphe. Lorsque cette fosse est trouvée, l'étape est complétée et la fosse est extraite du graphe. En 1995, Seymour a programmé l'algorithme de Vallet. Selon lui, cet algorithme convergera toujours vers la solution dans un nombre fini d'itérations. Cependant, avec un gisement de grande

dimension, le temps de résolution peut être très long. Pour atténuer cet inconvénient, il a proposé deux méthodes: 1) on pose initialement une grande teneur de coupure de façon à éliminer des calculs les branches vraiment trop faibles, 2) on procède à une agglomération des blocs.

3.2.3.2 La paramétrisation des réserves

La deuxième façon d'utiliser l'analyse paramétrique a été proposée par Matheron (1975) vers le milieu des années 70. Cette méthode a été développée dans le but d'éliminer complètement les paramètres économiques dans les données initiales. Les principales raisons mentionnées pour justifier un tel choix sont : 1) les coûts reliés à l'exploitation et au traitement du minerai varient en fonction de la grosseur du projet et du taux de production et ils ne peuvent donc pas être déterminés de façon précise avant que les contours optimaux ne soient eux-mêmes déterminés, 2) une solution n'est optimale que pour l'environnement économique qui a préalablement été fixé et 3) on suppose que les caractéristiques du gisement sont parfaitement connues alors qu'on se base en fait sur un estimé des réserves.

Ainsi, avec le principe de la paramétrisation des réserves, on cherche à avoir une valeur qui représente le contenu en minerai et la quantité de métal récupérable par rapport à un volume total extrait (exprimé en tonnes) plutôt que d'avoir une valeur économique associée à chacun des blocs d'un gisement. De cette façon, on élimine tous les paramètres économiques qui pourraient entrer en jeu et biaiser les résultats. L'article de Coléou (1988) offre un très bon aperçu de ce qu'est la paramétrisation des réserves.

L'objectif est le même que dans la paramétrisation, soit d'extraire le minerai le plus riche au début de l'exploitation de façon à maximiser la valeur présente nette du gisement. Après Matheron, plusieurs auteurs ont tenté de résoudre le problème des contours ultimes et des séquences de production à l'aide de cette méthode, notamment François-Bongarçon (1978), François-Bongarçon et Maréchal (1976-1977), François-Bongarçon et Guibal (1980-1984) et Coléou (1985-1987-1988).

Ainsi on voudra former une série de *l* fosses imbriquées :

$P_1 \subseteq P_2 \subseteq P_3 \subseteq \ldots P_l$ tel que $Q_1/V_1 > Q_2/V_2 > Q_3/V_3 > \ldots Q_l/V_l$

où Q_i = le contenu total en minerai compris dans la fosse P_i

V_i = le volume total de la fosse F_i

La façon d'obtenir les fosses imbriquées selon leur méthode est d'utiliser des fonctions paramétriques. Il s'agit en fait d'une heuristique qui permet de générer toutes les fosses lors d'une seule et même itération, comparativement à la paramétrisation qui doit utiliser Lerchs-Grossmann, ou toute autre méthode valable, pour déterminer chacune des fosses. Une fois la série de fosses trouvée, les paramètres économiques peuvent être à nouveau inclus dans le modèle pour déterminer les contours ultimes économiques et une séquence de production satisfaisante. Le logiciel MULTIPIT utilise la paramétrisation des réserves pour faire les calculs des contours ultimes et des séquences de production. Coléou (1988) affirme qu'il est plus rapide de générer une série de fosses à l'aide de MULTIPIT que d'appliquer l'algorithme de Lerchs-Grossmann à répétition comme le fait WHITTLE.

3.2.3.3 Le problème d'écart

Il arrive parfois, lorsqu'on applique la paramétrisation, que l'écart en volume entre deux fosses qui se suivent soit tellement grand qu'il est impossible de s'en servir lorsque vient le temps de produire une séquence de production. On appelle communément ce phénomène «le problème d'écart».

Wang et Sevim (1992-1993-1995) ont proposé une heuristique qui peut fournir une série de fosses imbriquées en éliminant le problème d'écart . L'algorithme détermine une série de fosses imbriquées telle que le nombre de blocs entre deux fosses consécutives ne dépasse pas une valeur fixée. En fait, ils ont proposé une version modifiée de l'algorithme de Gershon. Tout comme ce dernier, Wang et Sevim utilisent un cône

inversé lors de leurs calculs, à la seule différence qu'ils n'ont pas besoin de déterminer les contours ultimes de la fosse.

L'algorithme commence par déterminer la plus grosse fosse possible en éliminant tous les blocs qui ne pourront jamais faire partie de la fosse à cause des contraintes de pentes. Puis en partant du plus bas niveau, il détermine des cône inversés et calcule leur teneur moyenne. L'idée est d'éliminer des sous-groupes de blocs dont la teneur est la plus basse et ce, sans violer les contraintes de pentes; ainsi, on obtient la fosse qui contient la plus grande quantité de métal parmi toutes les fosses de même grandeur possibles.

Selon Thomas (1996), l'heuristique de Wang et Sevim souffre des limitations suivantes : 1) tout comme la méthode du cône mobile initiale, leur algorithme n'est pas capable de prendre en considération l'effet des cônes qui se superposent, 2) parce que les fosses imbriquées sont générés dans l'ordre inverse auquel ils vont être extraits (de bas en haut plutôt que de haut en bas), il sera très difficile d'incorporer d'autres contraintes à l'intérieur de l'algorithme, comme celles associées aux routes de halage. Toutefois, il semblerait que cette façon d'obtenir des fosses imbriquées se soit avérée concluante. Les auteurs affirment qu'ils sont en mesure d'obtenir une série de fosses qui est très près de celle fournie par la paramétrisation à l'aide de l'algorithme de Lerchs et Grossmann et ce, dans un temps très acceptable et sans avoir le problème d'écart.

3.2.4 La programmation dynamique

Plusieurs auteurs, tout comme pour le problème des contours ultimes, ont tenté de résoudre le problème des séquences de production à l'aide de la programmation dynamique et ce, sans beaucoup de succès. En fait, on classifie le problème des séquences de production comme étant un problème où la combinatoire est très élevée ; ainsi, la taille du problème augmente rapidement avec l'augmentation du nombre de blocs.

Roman (1974) fut le premier à proposer la résolution de ce problème à l'aide de la programmation dynamique. Il propose de trouver une séquence d'extraction pour tous les blocs puis à la fin, de déterminer les contours ultimes en fonction de la valeur présente nette. En 1974, il pouvait résoudre des problèmes de l'ordre de 2000 à 10 000 blocs, ce qui est en fait très petit et pour cause : il faut énumérer toutes les solutions possibles. Même encore aujourd'hui, il serait impossible de résoudre des problèmes de séquence de production de cette façon.

Par la suite, Wright (1988) a proposé une méthode 2D pour trouver une séquence de production. La méthode de programmation dynamique qu'il utilise pour trouver les séquences de production est la même que celle présentée dans son article de 1987. Dans cet article, lorsque l'algorithme calcule les contours ultimes, il passe par différentes étapes consécutives qui peuvent en fait servir de fosses intermédiaires. Donc, on utilise ces fosses intermédiaires pour déterminer des séquences de production possibles et on choisit la meilleure. Ce processus est basé un peu sur le principe d'essais et erreurs.

Dowd et Onur (1992-1993) proposent à leur tour une méthode pour déterminer les séquences de production. Tout d'abord, ils déterminent un contour ultime à l'aide de l'algorithme modifié de Korobov (voir la section 3.1.2), puis ils déterminent l'emplacement de la route de halage à l'aide d'une méthode heuristique, ils ajustent les pentes de la fosse aux exigences de la route de halage et finalement, ils déterminent une séquence de minage à l'aide d'un algorithme de programmation dynamique. La partie qui requiert le plus de temps de calcul est la détermination des séquences de minage. Pour que le problème ne soit pas trop lourd à traiter et pour éliminer certaines combinaisons possibles de solutions, ils ont appliqué des contraintes nécessaires dans le modèle, telles que les contraintes de mélange et les contraintes d'espace de travail. Malheureusement, même si plusieurs contraintes sont incluses dans le modèle, les auteurs affirment que le temps de résolution demeure très élevé. En fait, l'algorithme des séquences de minage fait le cheminement suivant :

- Il choisit un premier bloc à extraire parmi tous ceux qui peuvent être extraits. Ce bloc doit être un bloc accessible, c'est-à-dire un bloc qui n'a plus de prédécesseurs, donc complètement à découvert.

- Il trouve toutes les solutions possibles d'extraction à partir de ce premier bloc à l'aide d'un algorithme de programmation dynamique.

- Il choisit la meilleure solution parmi toutes celles trouvées.

- Il recommence pour tous les blocs qui peuvent être extraits lors de la première itération.

Cette méthode, en plus d'être très lourde quant au temps de résolution, ne tient nullement compte de la valeur de l'argent dans le temps.

Sevim et Lei (1995-1996) se sont basés sur les travaux entrepris par Wang et Sevim (1992-1993-1995) pour déterminer une séquence de production. Wang et Sevim, si on se réfère à la section 3.2.3.3, ont proposé une méthode pour résoudre le problème d'écart de la paramétrisation. Donc, à partir des fosses imbriquées obtenues grâce à cet algorithme, Sevim et Lei ont développé un algorithme de programmation dynamique qui permet de trouver quelle combinaison de fosses produira la plus grande valeur présente nette. Donc, en combinant la paramétrisation et la programmation dynamique, les auteurs affirment qu'il ont réussi à résoudre le problème circulaire des séquences de production et des contours ultimes.

Tolwinski et Golosinski (1995), Seymour (1995) et Wang (1996) proposent tous des algorithmes qui fonctionnent sur le même principe que celui de Sevim et Lei. Tolwinski et Golosinski (1995) déterminent des contours ultimes à l'aide d'un algorithme modifié de Lerchs-Grossmann puis ils déterminent une série de fosses imbriquées à l'aide de Lerchs-Grossmann en procédant à une agrégation de blocs pour augmenter la vitesse de

résolution. Finalement, ils appliquent la programmation dynamique pour trouver la combinaison de fosses qui maximisera la valeur présente nette.

Seymour (1995) et Wang (1996), quant à eux, considèrent qu'ils ont déjà en main une bonne série de fosses imbriquées qui sont les plus prometteuses, peu importe la méthode utilisée pour les trouver. Puis à l'aide de la programmation dynamique, ils trouvent une combinaison de fosses qui maximisent la VPN. Pour accélérer la programmation, ils incluent des contraintes comme le font Dowd et Onur (1992-1993).

3.2.5 L'intelligence artificielle

Devant l'impossibilité des techniques actuelles à fournir de bonnes solutions pratiques au problème des séquences de production, quelques chercheurs se sont tournés vers les techniques d'intelligence artificielle. L'inconvénient majeur rattaché à l'utilisation de ces techniques est que les résultats obtenus ne sont généralement pas reproductibles d'une fois à l'autre. Même si elles sont en mesure de fournir de très bons résultats, il faudra parfois faire plusieurs analyses différentes afin de vraiment obtenir la meilleure solution possible. Trois méthodes utilisant l'intelligence artificielle ont été développées durant les années 90. Il s'agit d'un algorithme développé par Tolwinski et Underwood (1992), d'un algorithme génétique développé par Denby et Schofield (1994) et d'un autre, développé par Elevli (1995). Thomas (1996) présente un bon résumé de ces techniques.

La méthode de Tolwinski et Underwood (1992) combine les concepts de l'optimisation stochastique et des réseaux de neurones artificiels pour produire un algorithme qui estime l'évolution optimale d'une mine à ciel ouvert. Le but de leur approche est d'obtenir des chemins de grande valeur à travers le gisement. Un chemin est défini comme une progression d'une fosse possible à une autre. En échantillonnant seulement une petite portion de tous les chemins possibles, l'algorithme de Tolwinski et Underwood tente de conserver en mémoire les caractéristiques qui produisent des chemins de grande valeur et ce,lles qui produisent des chemins de petite valeur. Le

changement d'étape, ou la transition d'une fosse à une autre dans le chemin, est gouverné par une distribution de probabilités et basé sur le nombre de fois qu'une étape particulière se retrouve dans les chemins de grande valeur. L'algorithme tient compte de la valeur présente nette de l'argent, des contraintes de pentes, de la largeur des bancs de travail et des contraintes de mélange. Selon Thomas (1996), si ce n'est de sa nature stochastique, le principal désavantage de l'algorithme est qu'il est à la merci des effets de l'explosion combinatoire, c'est-à-dire que la taille du problème augmente rapidement avec l'augmentation du nombre de blocs.

Elevli, en 1995, propose une méthode qui combine à la fois la programmation dynamique et le concept de «machine learning-expert system», une méthode d'intelligence artificielle, pour trouver une séquence de minage qui maximise les profits (VPN) tout en satisfaisant les contraintes techniques et économiques. Il s'agit en fait de programmation dynamique, mais grâce aux techniques d'intelligence artificielle, les solutions possibles ne sont pas toutes explorées.

En 1994, Denby et Schofield ont présenté un algorithme génétique qui permettait de développer une séquence de production en 2D. Il s'agit en fait du même principe que l'algorithme génétique pour déterminer les contours ultimes sauf que, cette fois, on veut tenir compte de la valeur présente nette de l'argent, des contraintes de pentes, de la largeur des bancs de travail et des contraintes de mélange (voir la section 3.1.5). Puis en 1996, Denby, Schofield et Hunter ont proposé une extension en 3D du même algorithme et en ont fait un module qui pouvait être implanté à l'intérieur du logiciel de «design» VULCAN.

Thomas (1996) propose l'idée d'utiliser la méthode de «Simulated annealing» pour résoudre le problème des séquences de production, mais ne l'applique pas.

3.2.6 Les méthodes de décomposition

Les méthodes de décomposition sont utilisées pour des problèmes classés comme étant des problèmes de très grande taille. Cette méthode consiste à résoudre seulement une portion du problème et à générer au fur et à mesure, lorsque nécessaire, les parties non explorées du problème. Pour utiliser de telles techniques, il faut identifier la structure particulière du problème et utiliser cette structure particulière afin de décomposer le problème en sous-problèmes pour lesquels des techniques de résolution optimales et efficaces sont connues. Cette façon de procéder a déjà été utilisée avec succès dans plusieurs domaines, comme en transport aérien.

La première tentative pour appliquer une méthode de décomposition au problème de séquences de production a été présentée en 1968 par Johnson dans sa thèse de doctorat. Johnson s'est inspiré du principe de décomposition de Dantzig-Wolfe et a proposé de décomposer le problème de séquences de production multi-périodes en plusieurs sous-problèmes plus petits. En présentant le problème des séquences de production d'une mine à ciel ouvert sous cette forme, il a été le premier à identifier la structure particulière du problème pour une période, puis pour plusieurs périodes. Comme nous l'avons mentionné plus haut dans la section 3.1.5, Johnson a formulé le sous problème, qui consistait à trouver les contours ultimes pour différents prix de métal (paramétrisation) comme un problème de transport. Le sous problème fournit donc une solution entière au problème maître. Le problème maître, quant à lui, est formulé comme un programme linéaire qui doit rencontrer les contraintes de ressources et de capacité de production. Comme le problème maître n'est pas formulé comme un problème en nombres entiers, ce problème doit être inclus à l'intérieur d'un algorithme d'évaluation et de séparation. La recherche d'une solution entière passe par l'exploration d'un arbre de possibilités qui peut être énorme. Ainsi, la recherche d'une solution en nombres entiers devient très difficile dans un temps raisonnable, voire impossible.

Quelques années plus tard, une seconde tentative pour résoudre le problème des séquences de production à l'aide de méthodes de décomposition est faite. En tant que sous-traitant pour Asarco, qui cherchait un bon logiciel de planification, la firme System Control Incorporated (SCI) en Californie a développé un algorithme qui tente de résoudre le problème des séquences de production. Afin de trouver une solution au problème, SCI a décidé de considérer le problème une période à la fois puis par la suite, de combiner les solutions individuelles à l'aide de la programmation dynamique. Même si cet algorithme ne prend pas en considération l'effet des différentes périodes les unes par rapport aux autres, il utilise de façon efficace la structure particulière du problème pour une période. En fait, pour son programme (ORE), SCI a utilisé les algorithmes de Lerchs et Grossmann, Lipkewich et Borgman ainsi que celui de Johnson qu'elle a combiné afin de trouver une solution acceptable au problème. Davis et Williams (1973), Williams (1974) et Sims (1979) sont tous des auteurs qui traitent du logiciel ORE.

En 1986, Dagdelen et Johnson ont publié un article qui se base en partie sur ce que Johnson a réalisé en 1968, mais cette fois, les auteurs ont développé une méthode de décomposition basée sur les techniques de relaxation Lagrangienne plutôt que sur celles de Dantzig-Wolfe. À notre avis, cet algorithme qui a été présenté en 1986 est la méthode pour déterminer une séquence de production qui donne la solution la plus proche de l'optimalité, en considérant l'ampleur du problème et sa nature circulaire comme nous l'avons expliqué dans une section précédente. Comme la théorie qui s'y rattache est assez compliquée et qu'un survol ne donnerait pas grand chose pour la compréhension, on a décidé de présenter l'algorithme en détail.

Soit le problème des contours ultimes qui peut être présenté sous la forme suivante et que nous appellerons le problème P1 :

$$\text{Max} \sum_i C_i X_i$$

Sous les contraintes :

$$X_i \in \Gamma_i \qquad \forall i$$

Où i = indice des blocs;

C_i = la valeur du bloc i;

X_i = 1 si le bloc i est extrait;

= 0 sinon;

Γ_i = l'ensemble des prédécesseurs du bloc i.

Il faut rappeler que le problème des contours ultimes n'impose aucune contrainte de capacité. Les seules contraintes présentes sont les contraintes des pentes (prédécesseurs) qui sont incluses dans le graphe. Comme nous l'avons déjà mentionné, nous savons très bien résoudre cette structure particulière du problème, soit à l'aide de Lerchs et Grossmann, soit à l'aide d'un algorithme de flot maximum. Dans la décomposition du problème, les auteurs ont donc essayé de conserver cette structure particulière du problème.

Soit le problème des contours ultimes avec un tonnage fixe qui peut être présenté sous la forme suivante et que nous appellerons le problème P2:

$$\text{Max} \sum_i C_i X_i$$

Sous les contraintes :

$$\sum_i a_i X_i = b$$

$$X_i \in \Gamma_i \qquad \forall i$$

Où b = tonnage fixe qu'on désire extraire,

a_i = le tonnage du bloc i.

Notons que la seule différence avec P1 est l'ajout de la contrainte de capacité. Toutefois, avec ce nouveau problème, on perd la structure attrayante de la formulation précédente. C'est pourquoi les auteurs utilisent la relaxation lagrangienne afin d'éliminer dans l'ensemble des contraintes, la contrainte relative à la capacité. Ainsi, en

incorporant un multiplicateur de Lagrange et en envoyant la nouvelle contrainte à l'intérieur de la fonction objectif, on réussit à retrouver la structure initiale du problème. Le nouveau problème des contours ultimes avec un tonnage fixe, où la contrainte de capacité est relaxée, peut être présenté sous la forme suivante et que nous appellerons le problème P3:

$$\text{Max} \sum_i C_i X_i - \lambda \left(\sum_i a_i X_i - b \right)$$

Sous les contraintes :

$$X_i \in \Gamma_i \qquad \forall i$$

Où λ est appelé le multiplicateur de Lagrange.

Compte tenu que λb est une constante, la présence de ce terme dans l'objectif n'est pas nécessaire. Après simplifications, on obtient la fonction objectif suivante :

$$\text{Max} \sum_i (C_i - \lambda a_i) X_i$$

On remarque que le problème P3 est identique au problème P1 à l'exception des coefficients des coûts. Donc, résoudre ce problème est équivalent à résoudre le problème P1 mais pour une valeur des blocs plus petite si $\lambda > 0$. Donc, plus λ est grand, plus la valeur des blocs diminue, moins on extrait de blocs et plus la fosse optimale est petite. Résoudre le problème P3 pour différentes valeur de λ revient à faire de la paramétrisation (voir la section 3.2.3).

Maintenant, nous verrons de quelle façon on peut utiliser le concept de paramétrisation pour trouver une séquence de production optimale (multi-périodes). Toutefois, ce problème est beaucoup plus compliqué que le problème pour une seule période (P3). Pour chacune des périodes, on doit extraire une certaine quantité de matériel. De plus, on ne peut plus seulement maximiser la somme des blocs extraits, on doit prendre en considération la valeur des blocs dans le temps. Donc pour différentes périodes, le

même bloc aura des valeurs différentes selon la période durant laquelle il est extrait. D'autre part, en plus des contraintes de prédécesseurs pour les pentes, on retrouve des contraintes de prédécesseurs pour les périodes. Ainsi, si un bloc est extrait dans la période 1, tous les blocs qui se trouvent dans le cône au-dessus de lui doivent nécessairement être extraits durant la période 1. Par contre, si un bloc est extrait durant la période 2, alors tous les blocs qui se trouvent dans son cône d'extraction doivent être extraits lors de la période 1 ou 2 et ainsi de suite. Soit le problème multi-périodes formulé en équations (P4) :

$$\text{Max} \sum_{p} \sum_{i} \left(C_i^p - \lambda_p a_i \right) X_i^p$$

Sous les contraintes :

$$X_i^p \in \Gamma_i^p \qquad \forall\, i, p$$

Où p = indice des périodes;

i = indice des blocs;

X_i^p = 1 si le bloc i est extrait à la période p;
= 0 sinon;

Γ_i^p = l'ensemble des blocs qui doivent être extraits à la période p ou avant si le bloc i est extrait à la période p;

C_i^p = coûts du bloc i à la période p.

Afin de résoudre ce problème, Dagdelen propose de décomposer le problème multi-périodes en P sous-problèmes dont chacun correspond à une période. Pour tenir compte de l'interaction entre les périodes, il traite les sous-problèmes séquentiellement en ajustant les coûts pour le sous-problème suivant.

En 1991, Tachefine, dans le cadre de son mémoire de maitrise, observe que la façon de calculer les coûts modifiés dans cet algorithme ne mène pas toujours à la solution optimale et propose une modification au calcul des coûts modifiés. En 1993, Zhao et Kim publient un article où ils mentionnent que cet algorithme ne converge pas et ainsi ne peut pas permettre de trouver une séquence de production optimale. Tachefine en

1993, dans le cadre de sa thèse de doctorat, donne une autre façon de fixer les multiplicateurs de Lagrange de manière à obtenir une convergence plus rapide et établit un critère d'arrêt respectable. En 1999, Akaine et Dagdelen incorporent à l'algorithme précédent la détermination des teneurs de coupures en plus des taux de production, soit un pas de plus vers la résolution du problème circulaire.

CHAPITRE IV
LES ALGORITHMES DE FLOT MAXIMUM ET LEUR COMPLEXITÉ

Comme nous l'avons vu dans le chapitre précédent, il existe beaucoup d'algorithmes pour déterminer les contours ultimes d'une fosse et les séquences de production. Certains d'entre eux sont très bons, tandis que d'autres ne le sont pas du tout. Le chapitre précédent permet au lecteur de se faire une idée sur le sujet. Parmi tous les algorithmes proposés, nous croyons que l'algorithme de décomposition de Dagdelen est très prometteur et que des recherches dans ce sens pourraient être fructueuses.

Tachefine (1993) de même que Hochbaum et Chen (1998) ont apporté beaucoup de points à explorer dans cette direction. Si on se réfère aux articles en question, Tachefine mentionne que le temps de résolution de l'algorithme varie de façon linéaire en fonction du nombre de blocs et Hochbaum et Chen affirment que d'inverser le graphe dans certaines circonstances améliore considérablement le temps de résolution. On s'aperçoit donc que l'algorithme de flot maximum varie en fonction de certains facteurs. Nous avons donc choisi d'isoler ces différents facteurs et de voir lesquels sont les plus importants et de quelle façon il serait possible de rendre cet algorithme de flot maximum le plus efficace possible. Il s'agit en fait, ici, d'une analyse de la complexité de l'algorithme de flot maximum appliqué au problème particulier des contours ultimes.

Les problèmes de flot maximum peuvent être facilement formulés de la façon suivante : dans un graphe orienté, où chaque arc porte une capacité, on désire envoyer le plus de flot possible entre deux nœuds, c'est-à-dire un nœud source s et un nœud puits t, et ce, sans excéder la capacité d'un seul arc du réseau. La majorité des algorithmes de flot maximum dans un réseau peuvent être classés en deux types : l'algorithme des chemins augmentants et les algorithmes de «preflow-push».

Dans les sections qui suivent, nous allons décrire les deux types d'algorithmes et les différences entre chacun d'eux. Pour le lecteur qui voudrait avoir plus de détails sur ces algorithmes et leur complexité, l'ouvrage de Ahuja et al. (1993) serait une bonne référence.

Avant de commencer, nous établirons une notation qui sera valide tout au long de ce chapitre lorsqu'il sera question du problème des contours ultimes. Soit G(N, A) le graphe des prédécesseurs G où N = {1, 2,..., i, ..., n} représente l'ensemble des nœuds et où $A = \{(i, j) \mid i \in N, j \in \Gamma_i\}$ représente l'ensemble des arcs avec Γ_i représentant l'ensemble des successeurs de i. Soit $G(N, A)$, le graphe modifié de Picard (le graphe G(N, A) auquel on a ajouté les nœuds source et puits). Soit $N^+ = \{i \in N \mid c_i > 0\}$ et $N^- = \{i \in N \mid c_i \leq 0\}$, où c_i représente la valeur du bloc *i*. Le nouveau graphe $G(N, A)$ est défini par $N = N \cup \{s, t\}$ et par $A = A \cup A^+ \cup A^-$, où $A^+ = \{(s, i) \mid i \in N^+\}$ et $A^- = \{(i, t) \mid i \in N^-\}$. La capacité des arcs est égale à c_i pour tous les arcs $(s, i) \in N^+$ et à $|c_i|$ pour tous les arcs $(i, t) \in N^-$.

4.1 Les notions de base des algorithmes de flot maximum

Tout d'abord, avant de présenter les algorithmes de flot maximum en détail, nous devons introduire certains concepts de base: soit le concept de graphe résiduel et le concept de coupe. Ces notions sont essentielles afin de bien comprendre les descriptions des algorithmes qui vont suivre.

4.1.1 Graphe résiduel

Le concept de graphe résiduel joue un rôle prédominant dans le développement de tous les algorithmes de flot maximum dont nous allons parler dans ce chapitre. Afin de bien comprendre le concept, nous allons présenter les quelques définitions suivantes. Soit G(x) un graphe sur lequel circule un flot x.

La *capacité résiduelle* r_{ij} de n'importe quel arc $(i, j) \in A$ est le flot maximum additionnel qui peut être envoyé du nœud i au nœud j en utilisant les arcs (i, j) ou (j, i).

Pour un arc (i, j), la capacité résiduelle r_{ij} est égale à $u_{ij} - x_{ij}$, c'est-à-dire la capacité non utilisée de l'arc (i, j). Pour l'arc (j, i), la capacité résiduelle r_{ji} est égale à x_{ij} le flot courant sur l'arc (i, j). Cette dernière capacité résiduelle permet de redistribuer le flot lorsque de mauvaises décisions ont été prises.

Soit le *graphe résiduel*, noté $G^r(x)$, qui est associé au graphe G pour le flot courant x. Pour chaque arc (i, j) qui a un flot x_{ij} dans le graphe original G, le graphe résiduel $G^r(x)$ contient au moins deux arcs:

1. un arc (i, j) qui a une capacité résiduelle $r_{ij} = u_{ij} - x_{ij}$, et
2. un autre arc (j, i) qui a une capacité résiduelle $r_{ji} = x_{ij}$.

De plus, le graphe résiduel $G^r(x)$ contient seulement des arcs tels que $r_{ij} > 0$. La figure 4.1 nous donne un exemple de ce qu'est un graphe résiduel.

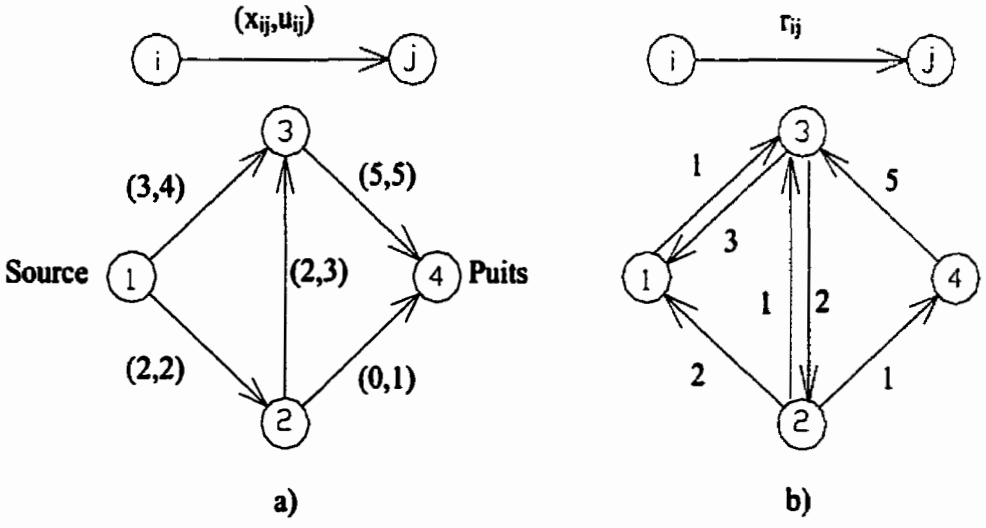


Figure 4.1 : Illustration d'un graphe résiduel : a) le graphe original G avec un flot x; b) le graphe résiduel $G^r(x)$

Comme l'ont mentionné Yegulap et Arias en 1992, en se basant sur cette définition de G^r, on remarque que les blocs d'air et les blocs de stérile qui ont une valeur nulle ne feront pas partie du graphe résiduel G^r étant donné que $r_{ij} = 0$ pour ces blocs. On en déduit donc que la construction seule de G^r va automatiquement réduire le graphe original.

4.1.2 Coupes dans un graphe

Nous allons maintenant donner quelques notions sur les coupes qui nous seront utiles plus tard.

Une *coupe* est une division de l'ensemble des nœuds N en deux sous-ensembles S et S' = N \{*s*}. On représente cette coupe en utilisant la notation [S, S']. De la même façon, on peut définir une coupe comme étant l'ensemble des arcs dont les nœuds initiaux et terminaux appartiennent à l'un ou l'autre des sous-ensembles S et S'.

On obtient une *coupe s-t* si $s \in S$ et $t \in S'$.

Un *arc «forward»* est un arc (i, j) où $i \in S$ et $j \in S'$ et un *arc «backward»* est un arc (i, j) où $i \in S'$ et $j \in S$ de la coupe [S, S']. Soit (S, S') l'ensemble des arcs «forward» de la coupe et (S', S) l'ensemble des arcs «backward» de la coupe.

On définit la *capacité* u[S, S'] d'une coupe *s-t*, [S, S'], comme étant la somme des capacités de tous les arcs «forward» appartenant à la coupe. C'est-à-dire :

$$u[S,S'] = \sum_{(i,j)\in(S,S')} u_{ij}$$

De façon plus précise, la capacité de la coupe est une borne supérieure de la quantité de flot maximum qui peut être envoyée des nœuds du sous-ensemble S vers les nœuds du sous-ensemble S'.

Une coupe *s-t* dont la capacité est minimum parmi toutes les coupes *s-t* possibles est une *coupe minimum.*

On définit la *capacité résiduelle* r[S, S'] d'une coupe *s-t*, [S, S'], comme étant la somme des capacités résiduelles de tous les arcs «forward» appartenant à la coupe, c'est-à-dire :

$$r[S,S'] = \sum_{(i,j)\in(S,S')} r_{ij}$$

Maintenant que nous avons présenté ces différentes notions, nous avons tous les outils nécessaires afin de bien comprendre les différents algorithmes qui seront présentés dans les sections suivantes et les notions théoriques qui s'y rattachent.

4.2 L'algorithme de chemins augmentants

Dans cette section, nous présenterons en première partie l'algorithme des chemins augmentants, puis l'algorithme d'étiquetage. En deuxième partie, nous définirons la complexité de l'algorithme d'étiquetage pour déterminer le chemin augmentant puis en troisième partie, nous présenterons les modifications qui ont été apportées au cours des ans afin d'améliorer l'algorithme initial et enfin, les hypothèses qui peuvent être faites par rapport au problème des contours ultimes en quatrième partie.

4.2.1 Description de l'algorithme

L'algorithme des chemins augmentants est l'un des algorithmes les plus simples et les plus intuitifs pour résoudre les problèmes de flot maximum. Avant de continuer, on doit spécifier certains termes.

Un *chemin augmentant* est un chemin direct entre le nœud source et le nœud puits dans le graphe résiduel. La *capacité résiduelle d'un chemin augmentant* est la capacité résiduelle minimum parmi tous les arcs du chemin augmentant. Par exemple à la figure 4.1 b) de la section précédente, on remarque qu'il y a exactement 1 chemin augmentant,

soit 1-3-2-4 et que la capacité résiduelle de ce chemin augmentant est $\delta = \min \{r_{13}, r_{32}, r_{24}\} = \min \{1, 2, 1\} = 1$. De plus, on observe que la capacité δ d'un chemin augmentant est toujours positive. Par conséquent, aussitôt qu'un graphe possède un chemin augmentant, on peut faire circuler des unités de flot supplémentaires du nœud source vers le nœud puits. L'algorithme des chemins augmentants est principalement fondé sur cette simple observation. L'algorithme procède de façon à identifier les chemins augmentants sur le graphe, puis à augmenter le flot sur ces chemins jusqu'à ce qu'il n'y ait plus de chemin possible du nœud source vers le nœud puits. Plus précisément, à chaque itération, l'algorithme identifie un chemin P du nœud *s* au nœud *t* dans le graphe résiduel $G'(x)$, où $x = 0$ au départ. Lorsqu'un tel chemin est identifié, la deuxième étape consiste à augmenter le flot sur ce chemin par une valeur δ qui correspond à la capacité résiduelle du chemin augmentant. Ce nouveau flot modifiera le graphe résiduel $G'(x)$. Cette procédure est répétée jusqu'à ce qu'il n'y ait plus de chemin augmentant possible entre le nœud *s* et le nœud *t*. À ce point, la solution optimale est atteinte.

La figure 4.2 présente les étapes de l'algorithme du chemin augmentant sur un petit réseau. En a), on montre le graphe résiduel pour une valeur de flot qui est nulle. En b), on montre le graphe résiduel après avoir augmenté le flot de 4 unités le long du chemin 1-3-4. En c), on montre le graphe résiduel après avoir augmenté le flot de 1 unité le long du chemin 1-2-3-4 et en d), le graphe résiduel après avoir augmenté le flot de 1 unité le long du chemin 1-2-4. Après d), on se rend compte qu'il n'existe plus de chemin augmentant possible. On a donc atteint la solution optimale avec un flot maximum de 6 unités.

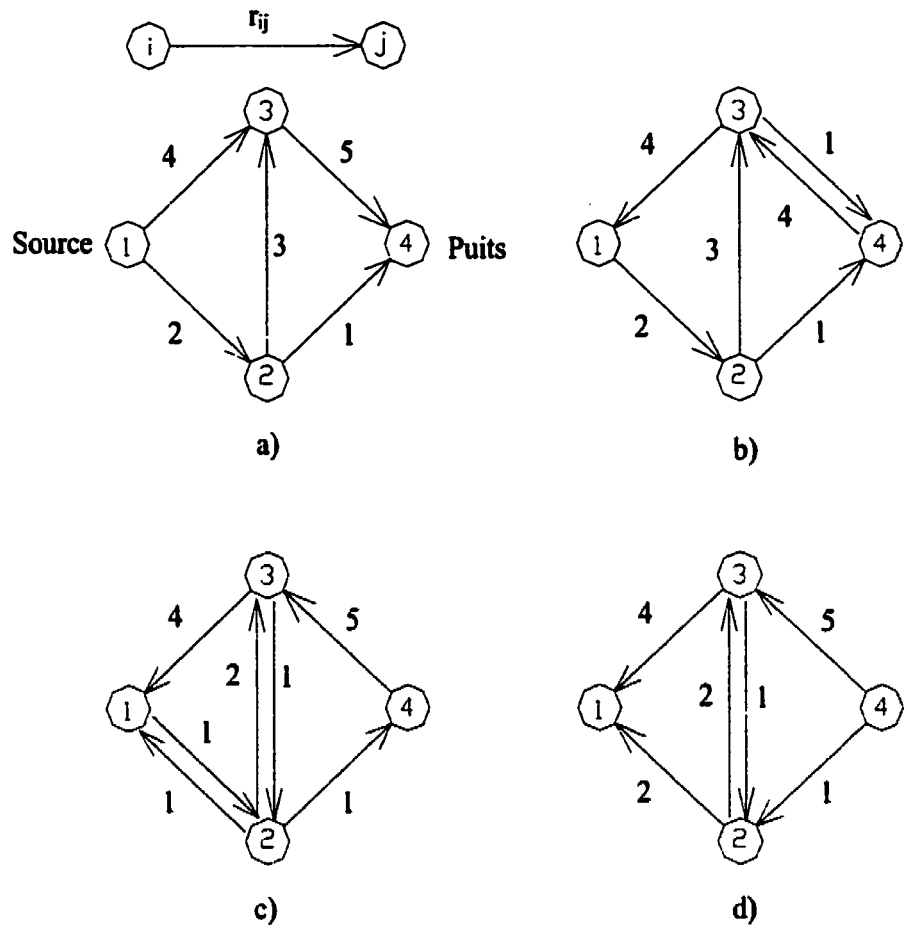


Figure 4.2 : Illustration de l'algorithme du chemin augmentant.

Jusqu'à maintenant, l'algorithme des chemins augmentants a été présenté de façon très générale. Nous n'avons pas encore préciser la façon dont le chemin augmentant est déterminé, ou tout simplement s'il en existe un ou pas, la façon de déterminer si l'algorithme va s'arrêter après un nombre fini d'itérations et quand il va s'arrêter, et la façon dont on détermine si on a atteint un flot maximum. En fait, toutes ces tâches sont effectuées par une sous-routine de l'algorithme des chemins augmentants appelée l'algorithme d'étiquetage. Cet algorithme est important parce que c'est de lui dont dépend la rapidité d'exécution des calculs.

L'algorithme d'étiquetage utilise une technique de recherche pour identifier un chemin direct entre s et t dans $G(x)$. Les algorithmes de recherche sont des techniques fondamentales en réseau pour trouver tous les nœuds dans un graphe qui satisfont certains critères. Selon les différentes variantes, ils représentent le cœur de plusieurs algorithmes de flot maximum et de flot à coût minimum.

Dans ce cas-ci, l'algorithme d'étiquetage se déploie à partir du nœud source pour trouver tous les nœuds qui peuvent être atteints à partir de ce dernier le long d'un chemin direct dans le graphe résiduel. À toutes les étapes, l'algorithme divise l'ensemble des nœuds du réseau en 2 groupes : ceux qui sont étiquetés et ceux qui ne le sont pas. Les nœuds étiquetés sont ceux que l'algorithme a atteins lors de l'opération de déploiement et les nœuds qui ne sont pas étiquetés sont ceux que l'algorithme n'a pas réussi à atteindre. La façon de fonctionner de l'algorithme est de choisir un nœud étiqueté et de suivre tous les arcs sortants de ce nœud de façon à étiqueter tous les nœuds possibles. Éventuellement, après un certain nombre d'itérations, le nœud puits est étiqueté. À ce moment, l'algorithme envoie le plus de flot possible à travers le chemin qu'il a identifié de s vers t. Les étiquettes sont ensuite effacées et le processus recommence. L'algorithme se termine lorsqu'à partir de tous les nœuds étiquetés il devient impossible de rejoindre le nœud puits. C'est cette sous-routine qui est au cœur de l'algorithme des chemins augmentants et c'est aussi elle qui va fixer le temps de résolution de l'algorithme. Dans la section qui suit, on donne un ordre de grandeur de ce temps de résolution.

4.2.2 La complexité de l'algorithme d'étiquetage

L'algorithme des chemins augmentants est exécuté dans un temps de l'ordre de O(mnU) où U représente la borne supérieure de la capacité des arcs du graphe. Cette complexité est composée de deux termes. Le premier est la recherche pour un chemin et le deuxième est le nombre de fois que cette recherche doit se produire, c'est-à-dire le nombre d'augmentations de flot.

On se rappelle que pour chaque itération, excepté la dernière, l'algorithme procède à une augmentation lorsque le nœud puits ne peut pas être étiqueté. Ainsi, il est aisé de voir que chaque augmentation requiert un temps d'exécution de O(m) parce que la méthode de recherche des chemins examine au plus une fois tous les arcs ou tous les nœuds. Ainsi, la complexité de l'algorithme est O(m) fois le nombre d'augmentations.

Mais quel sera le nombre d'augmentations? Si toutes les capacités sur les arcs sont intégrales (entières) et bornées par un nombre fini U, alors la capacité de la coupe (s, N \ {s}) est d'au plus nU, où «n» représente le nombre de nœuds. Par conséquent, la valeur maximale du flot est bornée par nU. Pour chaque augmentation, le flot sera augmenté d'au moins 1 unité. Par conséquent, l'algorithme s'arrêtera à l'intérieur de O(nU) augmentations, donc on peut affirmer que l'algorithme de chemins augmentants a une complexité de O(nmU).

Comme la complexité de l'algorithme est basée sur le pire cas, nous sommes en droit de nous demander qu'elle est la véritable complexité de l'algorithme lorsque celui-ci est appliqué au problème particulier des contours ultimes d'une mine à ciel ouvert. À première vue, cette complexité peut sembler désastreuse étant donné que la capacité des arcs $(i, j) \in A$ est très grande (théoriquement, les arcs du graphe des prédécesseurs portent une capacité infinie). Toutefois ces arcs ne feront jamais partie de la coupe minimale, autrement nous aurions un problème non borné. Donc, pour les explications suivantes, nous allons définir U comme étant la borne supérieure de la valeur absolue des blocs du gisement, c'est-à-dire que la capacité maximale se trouve parmi les arcs qui quittent le nœuds s, $(s, i) \in A^+$, ou parmi les arcs qui entrent au nœud t, $(i, t) \in A^-$. En pratique, nous savons que le nombre de chemins augmentants sera borné par $O(\beta)$ où $\beta = \min \{\beta_1, \beta_2\}$ et où :

$$\beta_1 = \sum_{i \in N^+} c_i \qquad \text{et} \qquad \beta_2 = \sum_{i \in N^-} |c_i|$$

Comme $\beta \ll nU$ nous avons là une bien meilleure borne supérieure.

4.2.3 Les améliorations

L'algorithme des chemins augmentants présente deux restrictions majeures. Premièrement, la complexité de l'algorithme pour le pire cas est de O(nmU), ce qui est peu attrayant lorsque vient le temps de résoudre des problèmes avec de grandes capacités sur les arcs (U). Deuxièmement, d'un point de vue théorique, pour des problèmes avec des données de capacité irrationnelle, l'algorithme peut converger vers une solution non optimale (Ahuja et al. (1993)). Bien que cette dernière observation n'ait aucun lien avec le problème qui nous intéresse, ces limitations suggèrent néanmoins que l'algorithme n'est pas entièrement satisfaisant en théorie. Malheureusement, l'algorithme est très peu satisfaisant en pratique aussi. En effet, appliqué à des problèmes de très grande taille, il requiert beaucoup trop de temps de calcul.

Au cours des années, on a tenté de résoudre ces problèmes en apportant différentes modifications à l'algorithme initial.

L'une de ces stratégies est particulièrement intéressante de par les notions qu'elle introduit et qui sont reprises dans les algorithmes de type «preflow-push». Le principe de cette stratégie est de limiter le type de chemins augmentants qui peuvent être utilisés à chaque étape. La façon développée pour y arriver est d'augmenter le flot le long du plus court chemin entre la source et le puits dans le graphe résiduel, le plus court chemin étant le chemin direct dans le graphe résiduel qui contient le moins d'arcs. Cet algorithme est appelé «*Shortest augmenting path algorithm*».

L'algorithme du «Shortest augmenting path» commence par étiqueter les nœuds selon leur distance du nœud puits. À chaque itération, il trouve le plus court chemin dans le graphe résiduel, procède à une augmentation de flot sur ce chemin, fait une mise à jour du graphe résiduel et étiquette à nouveau certains nœuds.

On appelle le type d'algorithme d'étiquetage qui est utilisé ici «*distance label algorithm*». Cet algorithme étiquette les nœuds d'une valeur d(i) à l'aide d'une fonction de distance. L'étiquetage des nœuds se fait par une recherche des nœuds en appliquant le critère de «largeur d'abord» en partant du nœud puits sur le graphe résiduel (*reverse breadth-first search*), où

$$d(i) = \min \{d(j) + 1 \mid (i, j) \in \Gamma_i\}$$

Plus précisément, on dit qu'une fonction de distance est *valide* si elle satisfait aux deux conditions suivantes:

$$d(t) = 0 \qquad \text{et} \qquad d(i) \leq d(j) + 1.$$

Si la fonction de distance est valide, alors l'étiquette de distance d(i) représente une borne inférieure de la plus courte distance en termes de nombre d'arcs (orientés) entre le nœud i et le nœud t dans le graphe résiduel $G^r(x)$. Soit un arc (i, j) dans le graphe résiduel $G^r(x)$. On dit que cet arc est *admissible* s'il satisfait aux deux conditions suivantes:

$$r_{ij} > 0 \qquad \text{et} \qquad d(i) = d(j) + 1$$

Tous les autres arcs qui ne rencontrent pas ces conditions sont dit inadmissibles. On dit qu'un *chemin du nœud s au nœud t est admissible* s'il est entièrement constitué d'arcs admissibles. Soit P un chemin admissible. La première condition implique que P est un chemin augmentant et la deuxième condition implique que P contient x arcs, $d(s) = x$, c'est-à-dire que dans de telles conditions, l'étiquette d(s) représente une borne inférieure de la longueur du plus court chemin de s vers t dans le graphe résiduel $G^r(x)$. Ainsi, n'importe quel chemin P sera un chemin qui contient le moins d'arcs possible. De là, on peut conclure qu'un chemin admissible est un plus court chemin augmentant du nœud s au nœud t dans le graphe résiduel $G^r(x)$.

Ainsi, si on augmente le flot le long du plus court chemin, la longueur du plus court chemin à l'itération suivante reste la même ou augmente. Donc, à l'intérieur de m augmentations, la longueur du plus court chemin augmentera forcément. Or, on sait qu'aucun chemin direct de s vers t ne peut contenir plus de (n - 1) arcs, ce résultat garantit que le nombre d'augmentations est d'au plus (n - 1) m. Donc, si $d(s) \geq n$, le graphe résiduel $G^r(x)$ ne contient aucun chemin direct de s vers t. Nous venons de prouver que la solution optimale est atteinte lorsque $d(s) \geq n$ et que le plus long chemin qui sera rencontré lors des calculs sera d'au plus n arcs. Ainsi la complexité pour la recherche du chemin vient de diminuer de O(m) à O(n), et par le fait même la complexité du «Shortest augmenting path» est $O(n^2m)$.

En pratique, il a été démontré qu'il est possible d'arrêter l'algorithme avant qu'il n'atteigne $d(s) = n$. La façon d'y arriver est d'utiliser un tableau à n dimensions, appelé «*list*», qui va calculer le nombre de nœuds qui ont une étiquette égale à k, où k = 0, 1, …., n - 1 pour chaque itération. Si pour un $k < d(s)$ spécifique, la liste est vide ($list(k) = 0$), alors l'algorithme s'arrête parce qu'il n'existe aucun chemin admissible de s vers t dans le graphe résiduel. De plus, ce saut représente la coupe de capacité minimum dans le graphe, c'est-à-dire que tous les nœuds i dont $d(i) > k$ font partie de l'ensemble des nœuds contenant s, tandis que tous les nœuds j dont $d(j) < k$ font partie de l'ensemble des nœuds contenant t. Plus communément, on appelle ce saut ou cet écart un *gap*. Dans le problème particulier des contours ultimes, l'ensemble des nœuds contenant s représente les blocs qui seront extraits, ce qu'on appelle une fermeture.

Une autre stratégie pour diminuer le temps de résolution de l'algorithme du chemin augmentant est de faire des augmentations de flot les plus grandes possible. Comme nous l'avons vu précédemment, l'algorithme de chemins augmentants peut procéder à une grande quantité d'augmentations, chacune ayant une petite quantité de flot. Donc ici, on propose d'augmenter le flot le long d'un chemin qui procure une grande capacité résiduelle de façon à ce que le nombre d'augmentations demeure relativement petit. Le «*Maximum capacity augmenting path algorithm*» utilise cette idée. Il augmente toujours

le flot le long du chemin qui a la plus grande capacité résiduelle. La complexité de cet algorithme est de O(m log U) augmentations. Une variation de cet algorithme appelé le «*capacity scaling algorithm*» augmentera le flot le long d'un chemin qui présente une capacité résiduelle suffisamment grande mais non maximale. Ce dernier présente lui aussi une complexité de O(m log U) augmentations, mais est beaucoup plus simple à implanter. Bien qu'il s'agisse là d'une méthode intéressante, elle ne sera pas décrite plus en détails dans ce mémoire.

Les deux méthodes qui viennent d'être présentées dans cette section, si elles sont combinées, peuvent améliorer considérablement l'efficacité de l'algorithme de chemins augmentants.

4.2.4 Observations par rapport au problème des contours ultimes

Voyons maintenant quelles conséquences ont ces améliorations sur le problème qui nous intéresse ici, soit les contours ultimes d'une fosse.

Pour montrer de quelle façon le nouveau critère d'arrêt, $list(k) = 0$,améliore la solution du problème des contours ultimes en pratique, nous formulerons les trois propositions suivantes:

Proposition 1 : Un chemin P orienté dans $G^r(x)$ entre s et t doit contenir un arc (i, j) où $i \in N^+$ et $j \in N^-$.

Preuve : Étant donné que $\forall\, i \in N^+$, $\exists\, (s, i)$ et $\nexists\, (i, t)$, que $\forall\, j \in N^- \exists\, (j, t)$ et $\nexists\, (s, j)$, alors le chemin entre s et t doit contenir au moins un arc (i, j) où $i \in N^+$ et $j \in N^-$.

Proposition 2 : $\forall\, i \in N^-$ l'étiquette de distance est initialisée à $d(i) = 1$.

Preuve : À partir de la définition de N^- on sait que tous les nœuds $i \in N^-$ sont reliés avec le nœud puits par un arc (i, j) qui implique que $d(i) = 1$.

Proposition 3 : S'il existe un chemin P de s vers t, alors l'étiquette de distance est initialisée à $d(s) = 3$.

Preuve : À partir du fait que le nœud s est relié seulement aux nœuds $i \in N^+$, tous les chemins P contiennent au moins un arc (s, i). Compte tenu que seulement les nœuds $j \in N^-$ sont reliés au nœud puits dans le graphe, alors tous les chemins P contiennent au moins un arc (j, t). Finalement, étant donné qu'il existe toujours au moins un arc (i, j) dans le graphe où $i \in N^+$ et $j \in N^-$, alors il est possible de construire un chemin s i j t dans le graphe, ce qui implique que $d(s) = 3$.

Étant donné que $d(s) = 3$ au début des calculs et que $3 << n$, il est facile de voir qu'il aurait fallu beaucoup d'opérations d'étiquetage avant d'atteindre le critère d'arrêt $d(s) = n$. Ainsi donc, l'utilisation d'un tableau «*list*» réduit le temps de résolution de l'identification de la coupe minimale, qui est en fait la seule chose qui nous intéresse vraiment dans la solution donnée par le flot maximum appliqué au problème des contours ultimes.

De plus, l'utilisation d'étiquetages de distance sur les nœuds permet implicitement de réduire le graphe. Pour expliquer cet aspect posons les propositions suivantes:

Proposition 4 : Soit l'ensemble $C \subseteq N^+$ une fermeture dans G. Alors $\forall\, i \in C$ il n'existe pas de chemin de s vers t.

Preuve : Si $i \in C$ alors tout $j \in \Gamma_i$ est lui aussi $\in C$, c'est-à-dire $\in N^+$. À partir de la définition de G, tous les nœuds $i \in C$ seront reliés au nœud s ou à tout autre nœud dans C. Maintenant, à partir de la proposition 1 nous pouvons conclure.

Si l'ensemble C représente une fermeture avec un poids positif, alors tous les blocs inclus dans C seront extraits, ce qui veut dire que tous les nœuds $i \in C$ feront partie du même ensemble que le nœud s dans la coupe minimum.

Proposition 5 : $\forall i \in C$, $d(i)$ peut être posé égal à n.

Preuve : À partir de la proposition 4, il est possible d'affirmer que lors du processus d'étiquetage, ces nœuds ne recevront pas d'étiquette compte tenu qu'il est impossible de rejoindre le nœud t par un chemin direct à partir de ces nœuds. Du moment où ils font partie de la fermeture de poids positif, chaque nœud $i \in C$ devrait être extrait, c'est-à-dire que leur étiquette à la fin du processus sera $d(i) \geq n$. Donc, en posant leur étiquette à n, ils ne seront pas considérés lors des calculs, sauf à la toute fin.

À partir de cette dernière proposition, on remarque que toute tentative de réduire le graphe en enlevant les blocs de poids positif des niveaux supérieurs (l'ensemble des blocs qui forment la fermeture) ne réduira pas les temps de calcul pour les algorithmes qui utilisent une sous-routine de «distance labels» étant donné que ces nœuds seront fixés à n lors du «pré-traitement».

4.3 Les algorithmes «preflow-push»

C'est dans l'espoir de développer des algorithmes dont la complexité dans les pires cas et dont les comportements empiriques seraient améliorés que les algorithmes plus puissants de «preflow-push» ont été développés au cours des dernières années. Il s'agit là d'une tout autre génération d'algorithmes de flot maximum. Ils sont non seulement plus efficaces quant aux temps de calculs, mais aussi plus solides pour résoudre tous les genres de problèmes de flot maximum. Ce sont ces types d'algorithmes qui nous intéressent dans ce chapitre.

Dans cette section, nous présenterons en première partie l'algorithme générique du «preflow-push». En deuxième partie, nous définirons la complexité de cet algorithme. En troisième partie, nous présenterons les modifications qui ont été apportées afin d'améliorer l'algorithme initial et différentes formulations de l'algorithme. Enfin, en quatrième partie, nous présenterons les hypothèses qui peuvent être énoncées par rapport au problème des contours ultimes.

4.3.1 Description de l'algorithme

Le principal inconvénient de l'algorithme des chemins augmentants est le temps qu'il passe à envoyer le flot le long d'un chemin, lequel requiert O(n) temps dans le pire des cas. La stratégie suivie pour produire les algorithmes de type «preflow-push» est de relaxer la contrainte de conservation de flot à des étapes intermédiaires, et ainsi ne pas obliger que chaque changement de flot soit une augmentation qui part du nœud source et se termine au nœud puits. En résumé, ces algorithmes recherchent le plus court chemin augmentant comme pour le «Shortest augmenting path algorithm», mais n'envoient pas de flot de la source jusqu'au puits le long de ce chemin. Ils envoient plutôt le flot sur les arcs de façon individuelle, ce qui fait que la contrainte de conservation de flot aux nœuds est violée. À chaque itération, la solution n'est pas réalisable. La solution optimale est atteinte lorsque la contrainte de conservation de flot est respectée à tous les nœuds. Cette façon de procéder permet à ces types d'algorithmes d'obtenir des temps de résolution qu'un algorithme de chemins augmentants ne pourrait jamais atteindre.

L'action d'envoyer δ unités le long d'un chemin de x arcs peut être décomposée en x opérations de base qui consistent à envoyer δ unités de flot le long de chaque arc du chemin. Chacune de ces opérations de base représente un «*push*», ou une «*poussée*». Quant au terme «*preflow*», ou «*pré-flot*», il provient des solutions intermédiaires obtenues lorsque les contraintes de conservation de flot ne sont pas respectées. De façon plus formelle, un «*preflow*» est une fonction de flot x qui satisfait les contraintes suivantes:

$$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in A$$

$$\sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} \ge 0 \qquad \forall i \in N$$

La dernière contrainte représente en fait la relaxation de la contrainte de conservation de flot qui aurait dû s'écrire:

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \qquad \text{où } b(i) = 0 \qquad \forall i \in N$$

En premier lieu, il y a une étape de «pré-traitement» qui consiste à pousser le plus de flot possible sur les arcs qui quittent la source. La quantité de flot sur chacun de ces arcs équivaut à la capacité de l'arc correspondant. Cette procédure résulte en un excès de flot, noté e(*i*), à chaque nœud $i \in N^+$. La contrainte de conservation de flot n'est donc par respectée parce que le flot qui entre aux nœuds *i* est égal à c_i alors que le flot qui en sort *i* est nul. Pendant la durée des calculs et à toutes les étapes intermédiaires, tous les nœuds, sauf le nœud source, seront soit en excès ($e(i) > 0$), soit équilibrés ($e(i) = 0$). Pour un «preflow» x, on définit l'excès à chaque nœud $i \in N$ comme étant :

$$e(i) = \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij}$$

Dans un «preflow», $e(i) \geq 0$ pour tous les nœuds $i \in N - \{s, t\}$. De plus, parce qu'aucun arc ne sort du nœud puits dans l'algorithme du «preflow-push», il existe un excès $e(t) \geq 0$ pour ce nœud. À partir du même raisonnement, on conclut que le nœud *s* est le seul nœud qui a un excès négatif. Les nœuds qui présentent un excès strictement positif sont appelés les nœuds actifs et, par convention, les nœuds *s* et *t* ne sont jamais considérés comme actifs. On notera l'ensemble des nœuds actifs, N^a. Après l'étape de pré-traitement, l'algorithme tente de pousser des unités de flot des nœuds actifs jusqu'au nœud puits.

Contrairement à l'algorithme de chemins augmentants qui maintient toujours une solution réalisable et qui tente d'obtenir une solution optimale, l'algorithme de «preflow-push» tente tout simplement d'obtenir une solution réalisable. Dans l'algorithme de «preflow-push», la présence d'un nœud actif indique que la solution est non réalisable. Conséquemment, l'opération de base de cet algorithme est de choisir un nœud actif et de tenter d'éliminer son excès en poussant le flot aux voisins. Étant donné que le but ultime

est d'envoyer le flot au nœud puits, ce dernier sera poussé vers le nœud le plus près de *t*. L'étiquetage de distance est utilisé pour déterminer ce nœud. Comme pour l'algorithme du «Shortest augmenting path», on mesure la proximité du nœud puits à l'aide des étiquettes de distance. Ainsi, envoyer un flot le plus près possible du nœud puits équivaut à pousser du flot sur un arc admissible.

Après l'étape de pré-traitement, le nœud source est immédiatement étiqueté d(s) = n. Une poussée de flot de δ unités sur un arc (i, j) admissible est appelé une *poussée saturée* si $\delta = r_{ij}$ et *non saturée* sinon. Lorsqu'une poussée non saturée est effectuée sur l'arc (i, j), le nœud i quitte l'ensemble N^a tandis que le nœud j y entre s'il n'y est pas déjà (si $j \notin N^a$). Si un nœud actif n'a pas d'arcs admissibles, alors son étiquette de distance est augmentée selon la fonction déjà mentionnée dans la section 4.2.3, soit :

$$d(i) = \min \{d(j) + 1 \mid (i, j) \in \Gamma_i\}$$

De cette façon, on crée au moins un arc admissible. Cette procédure permettra de ramener l'excédent de flot vers la source, c'est-à-dire que lorsque l'excès de flot ne pourra plus être envoyé vers le nœud puits, l'algorithme pourra ramener le flot excédentaire vers la source de façon à obtenir une solution réalisable, et l'algorithme se termine lorsqu'il n'y a plus de nœuds actifs. La procédure de retour de flot n'est pas essentielle dans plusieurs applications, et plus spécialement lorsqu'on n'est intéressé que par la coupe minimum comme c'est le cas pour les problèmes de contours ultimes.

4.3.2 La complexité de l'algorithme générique du «preflow-push»

Pour analyser la complexité de l'algorithme, nous devons tout d'abord établir le fait suivant: une étiquette de distance est toujours valide et n'augmente pas trop souvent. Cette conclusion vient des observation suivantes : 1) l'algorithme du plus court chemin augmentant maintient une étiquette de distance valide à chaque étape, 2) pour chaque étape d'étiquetage, l'étiquette de distance du nœud va strictement en augmentant, 3) à la façon de l'algorithme du «Shortest augmenting path», l'algorithme du «preflow-push»

pousse le flot seulement sur des arcs admissibles et ré-étiquette un nœud seulement lorsqu'il n'existe pas d'arc sortant qui soit admissible, et 4) à toutes les étapes de l'algorithme du «preflow-push», chaque nœud *i* avec un excès positif est connecté à la source par un chemin direct provenant du nœud *i* vers le nœud *s* dans le graphe résiduel.

Les étapes d'étiquetage qui changent l'étiquette d'un nœud consomment un temps de l'ordre de $O((n - 1)^2)$. Le nombre de poussées saturées est d'au plus O(nm) et le nombre de poussées non saturées est d'au plus $O(n^2m)$. Ainsi, l'algorithme générique du «preflow-push» consomme un temps de l'ordre de $O((n - 1)^2 + nm + n^2m) = O(n^2m)$, cette borne étant donnée par le nombre maximum de poussées non saturées qui seront effectuées. Nous verrons plus tard que cette borne n'est pas vraiment représentative de la réalité lorsque l'algorithme est appliqué au problème des contours ultimes.

4.3.3 Les améliorations

Comme nous venons de le voir, la complexité de l'algorithme générique du «preflow-push» est comparable à l'algorithme du «Shortest augmenting path». Toutefois, l'algorithme du «preflow-push» a quelques très bonnes propriétés, par exemple sa flexibilité et une grande place pour des améliorations ultérieures.

En spécifiant différentes règles pour le choix des nœuds actifs dans l'ensemble N^a pour l'étape de la poussée du flot et d'étiquetage, on peut dériver différents algorithmes, chacun ayant une complexité différente de la version initiale de l'algorithme générique du «preflow-push». Il existe plusieurs règles spécifiques pour examiner les nœuds actifs, les principales sont :

1. *FIFO preflow-push algorithm.* Cet algorithme examine les nœuds actifs selon le critère «First-in, first-out». Il présente une complexité de l'ordre de $O(n^3)$.

2. *LIFO preflow-push algorithm.* Cet algorithme examine les nœuds actifs selon le critère «Last-in, first-out».

3. *Highest-label preflow-push algorithm.* Cet algorithme pousse toujours le flot à partir d'un nœud actif qui possède l'étiquette de distance la plus élevée. Il présente une complexité de l'ordre de $O(n^2m^{1/2})$. On observe que cette complexité est meilleure que $O(n^3)$ pour tous les types de problèmes.

L'algorithme du «*FIFO preflow-push*» consiste à choisir les nœuds actifs l'un après l'autre dans l'ordre où ils ont été créés. Le premier nœud actif dans la liste N^a est choisi, puis éliminé de N^a. Si ce nœud possède un arc admissible (i, j), alors un flot de δ unités est poussé sur cet arc. S'il s'agit d'une poussée non saturée, alors le nœud i est éliminé de la liste N^a et le nœud j est ajouté à la fin de la liste. Si la poussée est saturée et que le nœud i est toujours actif, alors on conserve i et un autre arc admissible (i, j) doit être trouvé, puis j est ajouté à la fin de la liste N^a. Si on ne trouve pas d'arc admissible, alors le nœud i est étiqueté à nouveau et renvoyé à la fin de la liste N^a.

L'algorithme du «*LIFO preflow-push*» présente les caractéristiques suivantes. Soit i le nœud qui est considéré. Si une poussée non saturée est faite sur l'arc (i, j), alors le nœud i est retiré de la liste N^a et le nœud j est ajouté au début de la liste et devient ainsi le prochain nœud considéré. Si une poussée saturée est effectuée sur l'arc (i, j), alors le nœud j devient actif et, par le fait même, le nouveau nœud considéré est j et ce, même si i demeure actif.

La stratégie de *LIFO* peut être vue comme une recherche selon le critère de profondeur d'abord dans un graphe (si on considère que le nœud source est le nœud le moins profond du réseau), tandis que la stratégie du *FIFO* se rapproche plus d'une recherche selon le critère de largeur d'abord. La stratégie du *LIFO* tentera de pousser le flot à partir des nœuds actifs de l'ensemble N^a qui seront les plus près du puits. Considérant ce fait, il est difficile de croire qu'effectivement, les nœuds considérés englobent un intervalle plus grand, tel que mentionné par Hochbaum et Chen (1998). En fait, il serait plus plausible de dire qu'étant donné que l'algorithme utilisant la stratégie du *LIFO* travaille sur les nœuds qui ont la plus petite distance et que l'écart peut se produire à un

niveau k qui est plus près de 0, cette stratégie sera très efficace étant donné que tous les nœuds avec un d(i) > k seront étiquetés à d(i) = n. Par opposition, étant donné que la stratégie du *FIFO* travaille avec les nœuds actifs de différentes distances, elle ne produira pas nécessairement un écart aussi intéressant.

Ainsi que nous l'avons déjà mentionné dans la section précédente, l'opération qui limite l'algorithme générique du «preflow-push» est le nombre de poussées non saturées et les stratégies du *LIFO* et du *FIFO* qui viennent tout juste d'être présentées ne réduisent pas explicitement ce facteur. Toutefois, la troisième stratégie dont nous parlerons dans cette section, soit le «Highest-label», travaille directement sur cet aspect du problème, que nous venons de mentionner. La stratégie du *HL* consiste à ordonner les nœuds actifs en ordre croissant à partir de ceux dont l'étiquette est la plus élevée. En poussant le flot à partir des nœuds qui sont les plus éloignés du puits, il est possible éventuellement d'envoyer une plus grande quantité de flot à la fois sur les arcs. La principale idée derrière cette stratégie est vraiment de réduire le nombre de poussées non saturées sur les arcs. Il s'agit là, selon Ahuja et al., de l'algorithme le plus puissant à date pour résoudre les problèmes de flot maximum, justement à cause de cette propriété que nous venons tout juste de mentionner.

4.3.4 Observations par rapport au problème des contours ultimes

Hochbaum et Chen (1998) ont comparé les algorithmes du *LIFO «preflow-push»* et du *FIFO «preflow-push»* et ont obtenu de meilleurs temps de résolution à l'aide du *LIFO «preflow-push»* combiné à un algorithme de «gap-relabeling». L'algorithme de «gap-relabeling» étiquette les nœuds seulement lorsqu'il détecte un écart dans les étiquettes. Comme il a déjà été mentionné dans la section précédente, en utilisant la stratégie du *LIFO*, on augmente les chances qu'un écart se produise et qu'on identifie ainsi la coupe minimale plus rapidement. De cette façon, les poussées de «gap-relabeling» augmentent les étiquettes plus rapidement et diminuent le temps de résolution de l'algorithme.

Tachefine en 1997 a comparé l'algorithme du *FIFO* et celui du *HL* appliqué au problème des contours ultimes. Selon ses résultats, les meilleures solutions ont été obtenues à l'aide du *HL*. Tachefine a noté que la différence entre les temps de résolution de chacun des algorithmes augmentent en fonction du nombre de blocs, ou de la grosseur du graphe. De plus, pour les deux types d'algorithme, Tachefine note que le temps de résolution augmente de façon linéaire selon la grosseur du graphe, contrairement à ce qui est prédit par l'analyse de la complexité de l'algorithme $O(n^3)$. Étant donné que Tachefine a augmenté la grosseur de son graphe en augmentant aussi le nombre de niveaux, il est difficile de savoir si l'augmentation du temps de résolution est reliée à n, le nombre de blocs, ou à l, le nombre de niveaux, vu que le nombre de niveaux influence la distance maximale des chemins entre un nœud positif et le nœud puits (du moins au début des calculs).

Il serait très intéressant éventuellement de comparer les stratégies du *HL* et du *LIFO* combiné à un algorithme de «gap-relabeling» pour voir lequel des deux s'avérerait le plus prometteur. Toutefois, cette comparaison ne sera pas faite dans le cadre de ce mémoire.

Une autre stratégie d'accélération pour le problème des contours ultimes a été proposée par Hochbaum et Chen (1998). Elle consiste à résoudre le problème des contours ultimes en résolvant un flot maximum à l'aide d'un algorithme de type «preflow-push» sur un graphe inversé. Soit $G'(N', A')$ le graphe inversé correspondant à $G(N, A)$, où $N' = N$ et $A' = \{(i, j) \mid (j, i) \in A\}$. Avec cette stratégie, au lieu de pousser le flot à partir des nœuds positifs vers les nœuds négatifs, on fait l'inverse. Comme mentionné par les auteurs, lorsque la capacité de la coupe $[s, N \setminus s]$ est plus large que la coupe $[N \setminus t, t]$, alors la plus grande partie du flot utilisé contribue à garder les nœuds actifs et l'algorithme ne se termine pas tant que l'étiquette de ces nœuds n'atteint pas la valeur de n. En utilisant un graphe inversé dans de tels cas, ce qui devrait toujours se présenter dans les problèmes de contours ultimes, la plupart des poussées du puits vers la source

seront des poussées non saturées, ce qui aura pour effet immédiat de réduire rapidement le nombre de nœuds actifs de l'ensemble N^a et ainsi d'atteindre beaucoup plus rapidement le critère d'arrêt.

Comme pour l'algorithme du chemin augmentant, l'utilisation d'un algorithme d'étiquetage de distance permet de réduire implicitement le graphe, mais cette fois pour les nœuds ayant une valeur négative qui se trouvent au fond du gisement. Afin de prouver ce qui vient d'être avancé, posons $W = \{j \in N^- \mid \forall (i, j) \in G, i \in N^-\}$. Voici les 2 propositions suivantes:

Proposition 6 : Tous nœud j tel que $j \in W$ va toujours demeurer inactif tout au long des calculs.

Preuve : Aucune poussée ne sera effectué sur ces nœuds du moment où ils n'ont pas de prédécesseurs de poids positif. De cette façon leur excès restera toujours égal à 0, ce qui veux dire qu'ils ne feront jamais partie de l'ensemble des nœuds actifs N^a.

Proposition 7 : Dans la coupe de capacité minimale, tous les nœuds $j \in W$ vont se retrouver dans le même ensemble que t.

Preuve : Étant donné que ces nœuds ne pourrons jamais faire partie de l'ensemble N^a, ils ne seront jamais étiqueté à nouveau. Ainsi leur étiquette demeure égale à 1 jusqu'à la fin des calculs.

Ainsi, de la même façon que l'ensemble C représente une fermeture avec un poids positif, tel que présenté dans la section 4.2.4, l'ensemble W représente un ensemble de blocs de poids négatif localisé au fond du gisement. Tenter d'éliminer ces blocs lors du pré-traitement ne contribuera pas à sauver du temps lors des calculs étant donné qu'ils seront ignoré automatiquement. En résumé, on remarque que toute tentative de réduire le graphe en enlevant les blocs de poids positif des niveaux supérieurs (l'ensemble C) et les blocs de poids négatif des niveaux inférieurs (l'ensemble W) ne réduira pas les temps

de calcul pour les algorithmes qui utilisent une sous-routine de «distance labels» et un algorithme de type pré-flot.

CHAPITRE V
LE PROGRAMME UTILISÉ

Dans le chapitre précédent, la théorie concernant l'algorithme de flot maximum qui peut être appliqué au problème des contours ultimes a été décrite en détails. Tel que mentionné, il semblerait que le problème des contours ultimes, qui se présente toujours sous la même forme, ne représente pas le pire cas, mais qu'en fait, il varie en fonction de certains paramètres bien définis. Afin de prouver ce que nous avançons et aussi pour vérifier les dires de Tachefine (1997) et de Hochbaum et Chen (1998), nous voulions procéder à différents tests où nous pourrions faire varier différents paramètres de façon à cerner distinctement les facteurs qui influencent le plus la rapidité d'exécution de l'algorithme de flot maximum appliqué au problème des contours ultimes.

Pour ce faire, nous avions besoin d'un programme et de données qui nous permettraient d'effectuer des tests. Comme il n'existe pas de tels programmes sur le marché actuel, nous avons du créer notre propre programme. Ce chapitre présente les données qui ont été utilisées pour effectuer les tests ainsi que le programme qui a été créé à cette fin.

5.1 Les fichiers de données

Comme nous l'avons mentionné dans la section 2.3, les données nécessaires pour la détermination des contours ultimes se présentent sous forme d'un modèle de blocs pour lesquels on possède des informations sur la position et le contenu en minerai. Il existe deux alternatives possibles afin d'obtenir de tels fichiers qui nous permettraient de tester l'algorithme de flot maximum; soit qu'on génère des blocs de façon aléatoire, soit qu'on parte d'un modèle géologique réel.

5.1.1 Données réelles

Afin de bien tester l'algorithme, il était important pour nous d'avoir des données qui représentent un gisement réel. Nous avons donc réussi, grâce à la collaboration d'une mine à ciel ouvert[1], à avoir un fichier de données géologiques provenant de leur exploitation. Le fichier initial de type texte comporte 734 062 lignes où chacune représente un bloc du gisement et huit colonnes qui nous fournissent chacune une information sur le bloc en question. Ce fichier a été créé à l'aide du logiciel Surpac qui est un logiciel commercial de planification minière. Il permet entre autre de faire du «design» de fosse, de visualiser en trois dimensions le gisement et tous les éléments du design, de faire des patrons de forages, etc. Il s'agit d'un outil d'aide à la planification et au «design» très puissant, mais qui ne fait pas d'optimisation en tant que tel. En fait le logiciel Whittle dont nous avons brièvement parlé dans le chapitre III peut être greffé à Surpac pour permettre une optimisation. Il existe plusieurs autres logiciels commerciaux de ce genre qui ne diffèrent pas beaucoup l'un de l'autre, comme «Q-pit», «Vulcan» et «DataMine». Ils ont tous à peu près le même format, ils font tous à peu près les mêmes opérations et nous croyons qu'il ne serait pas si compliqué de changer d'un logiciel à l'autre. Dans le cadre de notre projet, nous avons choisi d'utiliser Surpac parce que ce logiciel était disponible.

La première colonne du fichier de la mine «M» contient le chiffre «1» pour tous les blocs. Il s'agit en fait d'un «string». Cette valeur est utilisée par le logiciel Surpac pour classer l'information. Les trois colonnes suivantes contiennent les coordonnées cartésiennes des blocs dans le gisement, soit y, x et z respectivement. Ici, il est bon de noter que le système de référence du gisement est positionné de façon à ce que l'axe des z augmente vers le haut. Cette information est très importante lors de la détermination des prédécesseurs pour la construction du graphe. La cinquième colonne contient des

---

[1] Nous appellerons cette mine «M» par souci de confidentialité.

valeurs entières qui sont appelées «litho». Dans cette colonne, on retrouve l'ensemble des valeurs suivantes : {19, 99, 100, 101, 102, 103, 104, 106, 108, 202, 203}. Tous les blocs du gisement ont une ou l'autre de ces valeurs dans cette colonne. En fait, elles servent à classifier les blocs. Si un bloc a une valeur 19 ou 99, il s'agit d'un bloc d'air. S'il a la valeur 103, il s'agit d'un bloc de minerai et s'il a une autre valeur, il s'agit d'un bloc de stérile. Les blocs d'air sont utilisés en modélisation afin de combler les vides formés par la topographie, de façon à obtenir un gisement rectangulaire. La sixième colonne donne la teneur en minerai du bloc. Pour l'air et le stérile, la valeur de teneur égale à une constante négative (-98). À la mine «M», on a seulement un type de minerai. La septième colonne contient une valeur de type réelle appelée «wrec» ou «weight recovery». Il s'agit en fait d'une valeur très proche de la teneur, mais un peu plus faible, qui représente la quantité qu'il est possible de récupérer lors du traitement. À la mine «M», on utilise cette valeur plutôt que la teneur dans les calculs des coûts. Finalement, la dernière colonne contient l'information concernant la densité du matériel. Le tableau 5.1 présente un résumé des informations sur le gisement.

Tableau 5.1 : Résumé des informations sur le gisement réel de 700 000 blocs

Type de Bloc	litho	Densité	Nombre de blocs
Minerai	103	Variable	44 430
Air	99	0	177 776
Air	19	0	1 164
Stérile	100	1.81	829
Stérile	101	2.2	10 816
Stérile	102	2.85	74 771
Stérile	104	2.65	301 657
Stérile	106	2.65	13 054
Stérile	108	2.75	84 207
Stérile	202	3.25	24 692
Stérile	203	2.8	666
		Total :	734 062

Si on regarde le tableau précédent, on observe que les blocs de minerai représentent seulement 6% du gisement. Les blocs ont des dimensions de 10 mètres de largeur sur 10 mètres de profondeur sur 14 mètres de hauteur (soit la hauteur des bancs). Nous appellerons donc ce format de fichier de données initiales, le format standard.

À présent, il est possible d'attribuer une valeur à chacun des blocs du gisement. Si nous avions voulu faire une étude plus réaliste, il aurait fallu inclure dans le calcul des coûts un facteur de réduction en fonction de la profondeur des blocs, car plus un bloc est profond, plus il devient coûteux de l'extraire parce qu'il nécessite plus de transport qu'un bloc situé plus près de la surface. Mais comme notre objectif n'est pas de faire une étude de faisabilité, nous avons décidé de simplifier les calculs des coûts à l'équation suivante :

Soit les termes suivants :

V_i = valeur associée au bloc i;
C_m = coûts de minage (dynamitage et transport dans la mine) pour le stérile et le minerai (à «M» ils sont de 1,80$/tonne);
C_c = coûts de traitement au concentrateur (à «M» ils sont de 4$/tonne de minerai);
C_t = coûts de transport (du concentré vers l'usine de transformation) ou d'expédition (à «M» ils sont de 3,50$/tonne de concentré);
P_v = prix de vente (à «M» il est d'environ 32$/tonne de concentré);
F_f = facteur de foisonnement[2], pour le transport vers l'usine de transformation (à «M» on utilise $F_f = 0.666$);
l = largeur des blocs;
p = profondeur des blocs;

---

[2] Le facteur de foisonnement est utilisé pour les calculs lors du transport des matériaux. Il tient compte des vides. Il s'agit en fait du rapport entre le volume ajouté d'une roche brisée reposant librement et son équivalent in-situ, non fragmenté.

h = hauteur des blocs;

D_i = densité du bloc i. Valeur fournie par le fichier initial;

W_i = %Wrec du bloc i. Valeur fournie par le fichier initial;

T_i = teneur en fer du bloc i si celle-ci est plus grande que la teneur de coupure, égale à 0 sinon. Valeur fournie par le fichier initial.

Si ($T_i > 0$, #1, #2).

Où #1 : $$V_i = D_i \times (l \times p \times h) \times \left[\left(W_i \times \left(P_v - \frac{C_t}{F_f} \right) \times T_i \right) - (C_m + C_c) \right]$$

#2 : $$V_i = -(D_i \times (l \times p \times h) \times C_m)$$

On doit lire cette équation comme suit : si la teneur en fer du bloc i est supérieure à 0, alors la valeur du bloc i sera calculée à partir de l'équation #1, sinon elle sera calculée à partir de l'équation #2. Il est important de spécifier que les teneurs qui nous ont été fournies dans le fichier de la mine «M» ont déjà été soumises à une teneur de coupure. Ainsi tous les blocs de stérile et les blocs d'air présentent une teneur nulle. C'est pourquoi on ne retrouve pas de blocs de minerai ayant une valeur voisine de 0 et que tous les blocs de stériles ont la même valeur négative. Il existe deux lignes de pensées pour le calcul des coûts sur les blocs. La première est telle que la mine «M» le fait. La deuxième est de ne pas appliquer de teneur de coupure et de calculer la valeur de tous les blocs à l'aide de l'équation #1. Si le résultats obtenu est plus petit ou égal à «0», alors on calcul les coûts à partir de l'équation #2. À mon avis la deuxième façon de faire est plus réaliste. De cette façon on élimine le principe de teneur de coupure des décisions et alors le logiciel d'optimisation fait vraiment tout le travail. Toutefois, en pratique, il n'est peut être pas possible de la mettre en application.

Lorsqu'on fait les calculs pour les 734 062 blocs, on observe que les valeurs varient entre 6 043 et 86 811 pour les blocs de minerai et qu'en moyenne, les blocs de stérile ont une valeur minimum de -8 189.

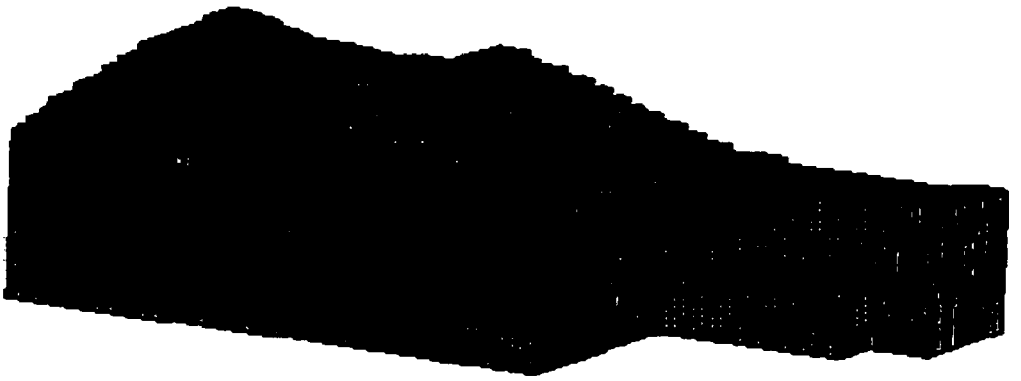


Figure 5.1 : Gisement de la mine «M» tel que vu à partir du logiciel Surpac

5.1.2 Données aléatoires

Afin d'évaluer la performance de l'algorithme qui sera utilisé et d'isoler les éléments susceptibles d'influencer les temps de résolution, plusieurs fichiers de données ont été créés en générant des données aléatoires. Il est important de noter que ces fichiers créés par des nombres aléatoires ne reflètent pas la réalité; toutefois ils permettent d'isoler et de contrôler plus aisément des éléments tels que le nombre de blocs par niveau, le nombre de niveaux par gisement, le ratio du nombre de blocs de minerai versus le nombre de blocs de stérile, etc. On dit que les gisements dont les coûts (teneur) ont été générés de façon aléatoire ne sont pas représentatifs de la réalité parce la distribution des teneurs dans ce cas sera très hétérogène alors que normalement, dans un gisement réel, les teneurs sont réparties de façon plus graduelle et plus homogène et sont concentrées dans des zones dites minéralisées.

Afin de conserver une similitude avec le gisement réel, les coûts ont été choisis aléatoirement dans l'intervalle {6 000, 86 000} pour les blocs de minerai alors que les coûts des blocs de stérile ont été fixés à -8 000.

5.2 Le programme

Comme nous avons pu le constater dans le chapitre II, certains chercheurs ont développé des logiciels qui calculent les contours ultimes à l'aide d'un flot maximum, mais il n'existe pas, à notre connaissance, d'équivalent dans les logiciels commerciaux. Ce fait nous force donc à construire notre propre programme basé sur un algorithme de flot maximum dans un graphe afin de pouvoir effectuer les tests qui nous intéressent.

Il existe plusieurs sites Internet portant sur la recherche opérationnelle où il est possible de se procurer différents algorithmes pour résoudre des problèmes communs. C'est dans l'un de ces sites que nous avons trouvé un algorithme de flot maximum de type «Highest Label Preflow» basé sur l'algorithme développé par Goldberg et Tarjan (1988)[3]. Tel que mentionné par Tachefine (1993) et comme nous l'avons vu dans le chapitre précédent, il s'agit là de l'algorithme le plus efficace pour résoudre notre problème. Cet algorithme, que nous avons déniché sue Internet, a été programmé en C, il fonctionne uniquement sur des stations en exploitation UNIX ou LINUX et il contient déjà une procédure qui lui permet de donner les temps de calcul pour le flot maximum seulement. Cette dernière caractéristique sera extrêmement pratique lors des tests. Éventuellement toutefois, il faudrait penser à modifier le code de l'algorithme du flot maximum de façon à pouvoir faire fonctionner le programme sur tous les types de stations.

Pour ce qui a trait au traitement de données nécessaire à la détermination des contours ultimes, comme le calcul des prédécesseurs, une programmation devra être faite. Le langage de programmation utilisé sera le C++.

---

[3] Nous nous sommes procuré le code du flot maximum à l'adresse Internet suivante : http://archi.snu.ac.kr/bwkim/free_or_codes.html, et nous avons utilisé l'implémentation Golberg 5 de l'algorithme.

5.2.1 Les objectifs à atteindre

Dans cette partie, nous définirons ce qu'il est légitime d'exiger d'un programme d'optimisation des séquences de production, c'est-à-dire ce que le logiciel idéal devrait faire. Ces caractéristiques sont définies par la connaissance que nous avons du problème, par ce qui a été fait par le passé et par ce dont nous avons besoin pour réaliser nos tests. Donc, certains des objectifs mentionnés dans cette section seront essentiels alors que d'autres seront optionnels, c'est-à-dire que même si ces objectifs ne sont pas intégrés dès maintenant dans le projet, ils n'influenceront en rien les résultats que nous désirons prouver ici et pourront être ajoutés plus tard afin de rendre la résolution du problème des séquences de production plus réaliste et applicable à de vraies mines.

Qu'il y ait un ou plusieurs types de minerai, que le système de référence soit vers le haut ou vers le bas, ou qu'il y ait plusieurs pentes selon les différentes zones, le programme devrait pouvoir s'appliquer à n'importe quel type de gisement. On voudrait que le programme soit le plus général possible.

On a besoin d'un programme qui peut lire un fichier sous la forme standard (voir la section 5.1.1) contenant environ 800 000 blocs et qui peut le réduire sur demande. Éventuellement, un bon programme commercial devrait pouvoir lire n'importe quel type de fichier d'entrées afin de le rendre plus flexible et plus facile d'utilisation. Quant à la fonction de réduction, elle va nous servir afin de créer des gisements de différentes grosseurs pour tester le flot maximum dans différentes conditions. Dans un logiciel commercial, il pourrait s'agir d'une option mais en fait il s'agit là d'une application plus ou moins pertinente lors de la résolution de cas réels. Par contre, cette fonction sera extrêmement utile pour réaliser les tests que nous devons faire. À partir des fichiers réduits, on devrait pouvoir effectuer l'optimisation des contours ultimes immédiatement ou créer un fichier qui contiendra le nombre voulu de blocs.

Comme les coûts sur les blocs ne sont pas fournis dans le fichier initial, on veut que le programme puisse les calculer. Cette étape pourrait être faite à l'aide du logiciel Surpac,

mais il est plus simple de la faire à l'intérieur du programme. Ainsi on se simplifie grandement la tâche si jamais il devient nécessaire de faire différents tests en faisant varier les coûts. Il est plus facile de changer seulement un prix que de refaire un fichier au complet sous la forme standard avec de nouveaux coûts calculés à l'aide de Surpac. Afin de calculer les coûts à l'intérieur du programme, l'usager devra fournir les informations concernant les prix et les coûts de production. On se rappelle qu'à la mine «M», nous avons seulement un type de minerai. Donc le calcul des coûts que nous aurons à faire est plutôt simpliste, surtout qu'il y a beaucoup de facteurs dont nous ne tiendrons pas compte. Mais il s'agit là, en fait, de la partie la plus importante et la plus compliquée de la définition d'un gisement. Un bon programme commercial devrait pouvoir simplifier cette tâche à l'utilisateur sans pour autant biaiser les valeurs qui seront attribuées aux blocs. De plus, rares sont les exploitations qui n'extraient qu'un seul type de minerai donc, éventuellement, un logiciel commercial devrait pouvoir permettre de calculer les coûts en tenant compte de cet état de fait.

Pour les seuls besoins des tests que nous devons effectuer, le programme que nous allons concevoir devra permettre de générer des coûts de façon aléatoire et d'inverser le graphe à la demande. Or, un programme commercial qui permettrait de générer des coûts de façon aléatoire ne serait d'aucune utilité dans l'industrie. Par contre, si les tests s'avèrent concluants, il serait probablement très avantageux d'inclure un module à l'intérieur d'un logiciel commercial qui permettrait d'inverser le graphe s'il y a possibilité d'amélioration du temps de résolution.

Pour notre programme et un programme commercial, il est très important que la quantité de variables (nombre de blocs) ne soit pas une limitation par rapport à l'espace mémoire de l'ordinateur. On parle ici de très, très grands problèmes. Il ne faut pas oublier que pour chacun des blocs, l'information suivante est connue et doit être emmagasinée en mémoire : les coordonnées, la «lithographie», la teneur, la densité, le «recouvrement poids» ou «wrec». Pour effectuer une paramétrisation, il faut alors conserver un coût pour chacune des périodes, garder en mémoire un réseau ayant un minimum de neuf

prédécesseurs par nœud et finalement, une fois que l'optimisation est terminée, il faut conserver pour chacun des blocs la période d'extraction, ce qui représente un nombre impressionnant de données à emmagasiner. De plus, afin de rendre la compréhension du programme aisée, on veut limiter le nombre de fichiers qui seront utilisés et créés.

Une fois ces différents points implantés, il ne sera pas très compliqué d'inclure un processus itératif qui permettra de faire de la paramétrisation. Bien entendu, tous les résultats obtenus devront pouvoir être visualisés en trois dimensions à l'aide du logiciel Surpac. Cette option nous permettra de vérifier si les solutions fournies sont réalisables de façon pratique ou non. Éventuellement, on voudrait que ce programme calcule des séquences de production à l'aide de l'algorithme développé par Dagdelen, mais avec les coûts modifiés tels que présentés par Tachefine; et qu'il y ait un module à l'intérieur qui permette l'optimisation d'une séquence de production en tenant compte des routes de halage. L'objectif principal est que le logiciel soit facile à comprendre et à utiliser et, surtout, qu'on puisse facilement lui apporter des améliorations ultérieures.

Évidemment, plusieurs points ont été oubliés ici, sauf que les plus importants, ceux qui font vraiment le travail d'optimisation, ont été mentionnés. Il est certain qu'un logiciel commercial, pour satisfaire un utilisateur, doit répondre à plusieurs besoins et offrir plusieurs alternatives comme : pouvoir déterminer le format des fichiers de données et des fichiers de sorties, déterminer le type d'optimisation et être beaucoup plus flexible. Mais souvent, il s'agit de détails qui peuvent facilement être incorporés par un programmeur chevronné.

5.2.2 Les objectifs atteints

Pendant ces deux années de maîtrise, un programme qui nous a permis de faire des tests sur l'algorithme de flot maximum a été développé. Pendant l'élaboration de ce programme, différents problèmes ont été rencontrés. Certains se sont réglés d'eux-mêmes et d'autres se trouvent hors de notre contrôle. Dans cette section, le travail de programmation qui a été fait lors de ces deux années sera présenté et discuté. Bien

entendu, le programme développé n'a pas toutes les caractéristiques d'un programme commercial, mais là n'était pas le but du travail.

Tout d'abord, il a fallu trouver une façon de gérer et d'emmagasiner toute l'information nécessaire aux calculs. Afin de résoudre le problème concernant la gestion des nombreuses informations pour chacun des blocs, la décision a été prise de mettre toute l'information dans un fichier binaire. L'avantage d'utiliser un fichier binaire est que l'information est rangée selon un type construit, ou plus facilement, sous forme de tableau. Ainsi l'information n'est pas conservée en mémoire. De plus les fichiers binaires permettent un accès direct à l'information, c'est-à-dire qu'on a pas à lire tout le fichier, ligne par ligne depuis le début, pour avoir de l'information sur le bloc no 66 401 par exemple. Si cette façon de procéder n'avait pas été employée, on n'aurait pu résoudre que des problèmes ne dépassant pas l'ordre de 100 000 blocs et encore, avec un bon ordinateur. Avec les fichiers binaires, on améliore beaucoup la capacité du programme à emmagasiner de l'information et à traiter des problèmes de grandes tailles.

Par contre, l'algorithme de flot maximum que nous avons trouvé sur Internet en a. Lorsque vient le temps de résoudre des problèmes de l'ordre de 500 000 blocs, un message d'erreur provenant du code du flot maximum nous dit que l'espace mémoire disponible a été dépassé. Nous n'avons pas trouvé la source de ce message. Donc, jusqu'à maintenant, des gisements rectangulaires jusqu'à environ 300 000 blocs ont pu être résolus. Nous n'avons toutefois pas testé les limites du flot maximum en termes de blocs. Ce que nous savons, c'est qu'elles se situent entre 300 000 blocs et 500 000 blocs, ce qui est néanmoins très considérable. Whittle spécifie, dans son manuel d'utilisateur, qu'il n'est pas vraiment pertinent de résoudre des problèmes qui contiennent plus de 200 000 blocs, étant donné le temps de résolution que les calculs prennent versus la différence dans les résultats. On connaît pas, par contre, le maximum de blocs que le logiciel peut traiter à l'aide d'un P.C. normal. Donc, nous considérons qu'avec une résolution possible de 300 000 blocs dans un temps raisonnable, nous sommes tout à fait

dans les normes. Tous les tests qui seront faits impliqueront des graphes de moins de 150 000 nœuds, soit des gisements qui contiennent initialement environ 270 000 blocs.

Un deuxième désavantage de l'algorithme du flot maximum est qu'il ne fonctionne qu'à partir de systèmes en exploitation UNIX ou LINUX. Donc éventuellement, il faudrait que l'algorithme du flot maximum choisi ne soit pas dépendant du système d'exploitation. Encore une fois, un bon programmeur pourrait résoudre ce problème. Présentement, le programme ne fonctionne que pour un gisement qui contient un seul type de minerai et il ne peut pas tenir compte de plusieurs zones ou plusieurs pentes. Éventuellement, il faudrait penser à pouvoir traiter plusieurs types de minerai et à inclure un patron de recherche conique pour tenir compte de différentes pentes, voir la section (3.1.3.1). Étant donné qu'il n'existe pas de module pour produire des pentes variables, pour le calcul des prédécesseurs, on a fixé les pentes selon un patron 1:9 pour tous les blocs. Donc, lors de la recherche des prédécesseurs du bloc i, les 9 blocs qui se trouvent directement au-dessus de ce dernier sont choisis. Bien sûr, le patron 1:9 a été choisi de façon tout a fait arbitraire. Il aurait été facile d'en considérer un tout autre, par exemple un patron 1:5 aurait très bien pu faire l'affaire. Il est important de noter que l'utilisation d'un patron 1:9 ne donne pas nécessairement des pentes à 45°. Les pentes dépendront des dimensions des blocs. Pour le gisement que nous avons, les pentes réelles seront d'environ 55°.

Le programme a été conçu de façon à être indépendant du système de référence. L'usager n'a qu'à entrer le sens de l'axe des «z» lorsqu'on le lui demande et le programme calcule les prédécesseurs automatiquement en fonction de cette réponse.

De plus, selon un intervalle de prix de vente et un nombre de périodes voulus, le programme est en mesure d'effectuer la résolution de l'algorithme de flot maximum à plusieurs reprises afin de faire de la paramétrisation telle que Whittle la fait. À chaque itération, il augmente le prix du métal d'un Δ donné, il recalcule les coûts, il récrit le fichier de données pour le flot maximum selon les nouveaux coûts et il fait une nouvelle

optimisation. Puis les blocs qui sont extraits lors de cette itération i et qui n'ont pas déjà été extraits lors d'une itération précédente se voient attribuer le numéro de l'itération i qui donne en fait la période d'extraction du bloc. Cette partie n'était pas vraiment essentielle pour faire les tests dont nous avions besoin, mais nous l'avons tout de même incluse dans le programme.

Le programme a besoin de plusieurs fichiers pour fonctionner, par contre l'usager n'en voit que deux : le fichier d'entrées et le fichier de sorties. Le fichier d'entrées est un fichier sous la forme standard, soit identique à celui décrit à la section 5.1.1. Le fichier de sorties est un fichier identique au premier, à l'exception de deux colonnes qui ont été ajoutées. Une première pour dire si le bloc fait partie du cône d'extraction ou non (graphe des prédécesseurs), et une deuxième qui dit si le bloc a été extrait ou non, et si oui, à quelle période. Un nombre «0» dans la première colonne signifie que le bloc fait partie du cône d'extraction et un nombre «1» signifie qu'il a été éliminé du cône d'extraction. Un bloc qui présente la valeur «0» dans la deuxième colonne ne sera jamais extrait. Par contre, si on y trouve une valeur plus grande que «0», le bloc sera extrait à la période correspondant à la valeur. Le format du fichier de sorties peut facilement être modifié. Ainsi, on aurait très bien pu donner seulement les coordonnées et dire à quelle période le bloc était extrait. Ce sont les valeurs de cette dernière colonne du fichier de sorties qui permettent de visualiser la fosse optimale en trois dimensions dans Surpac.

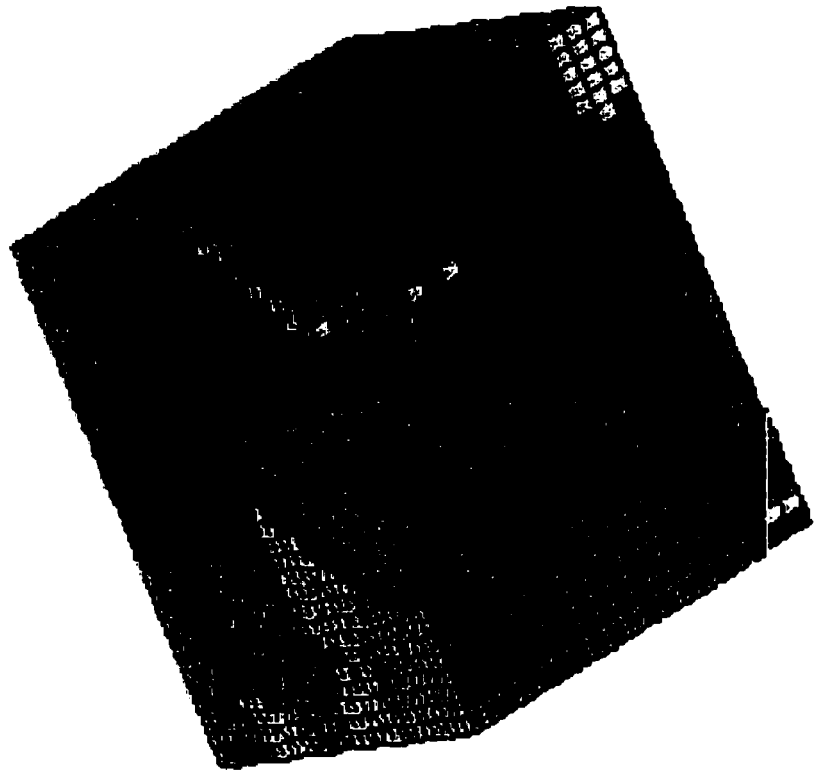


Figure 5.2 : Exemple d'un gisement rectangulaire de 20 000 blocs avant l'optimisation

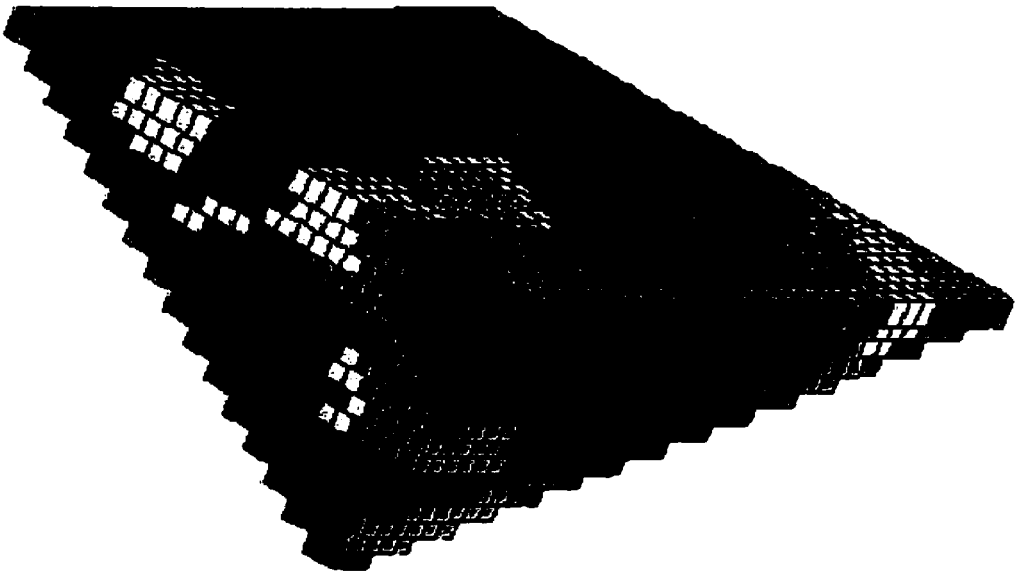


Figure 5.3 : Exemple d'un cône d'extraction tel que calculé à l'aide du programme à partir du gisement de 20 000 blocs

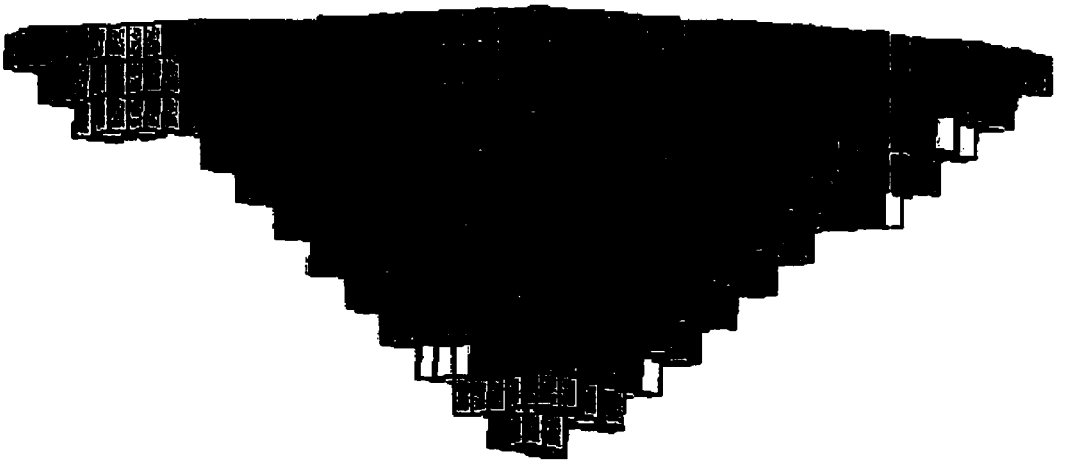


Figure 5.4 : Exemple d'une fosse optimale calculée à l'aide du programme pour le gisement de 20 000 blocs

Outre les fichiers d'entrées et de sorties, il y a un fichier binaire «data.dat» qui recueille toutes les informations sur chacun des blocs, un fichier texte «input.max» qui contient la structure du graphe pour le flot maximum et un fichier «coupe.out», créé par le flot maximum et qui contient le numéro des blocs qui sont extraits une fois l'optimisation terminée pour une période. Le programme peut aussi, à la demande de l'utilisateur, lire un fichier qui contient un certain nombre de blocs et le réduire de façon à avoir un fichier plus petit qui pourra être optimisé par la suite. À ce moment, c'est l'utilisateur qui spécifie le nom du fichier réduit.

Pour les tests, on a créé 4 programmes à partir du programme qui calcule les coûts pour un gisement réel. Les quatre programmes sont semblables à quelques exceptions près. Le premier programme, appelé «paramninv.cpp», lit un fichier d'entrées, peut le réduire, calcule les coûts des blocs à l'aide des paramètres fournis par l'utilisateur et calcule les contours ultimes à l'aide d'un flot maximum pour une ou plusieurs périodes. Le programme «paraminv.cpp» est exactement le même que le précédent, sauf qu'on inverse le graphe. Ensuite le programme «randomninv.cpp» est encore le même que le premier, sauf que cette fois, il génère des coûts associés à chacun des blocs de façon aléatoire selon un ratio minerai/stérile spécifié par l'utilisateur. Finalement le programme «randominv.cpp» en plus de générer des coûts de façon aléatoire, inverse de graphe avant de résoudre le problème de flot maximum. À l'aide de ces quatre programmes, il sera possible de faire tous les tests nécessaires à une meilleure compréhension de l'algorithme du flot maximum appliqué au problème particulier des contours ultimes. L'annexe III présente le listing complet de «paramninv.cpp» et les modifications apportées pour les trois autres programmes.

5.3 Fichiers tests

À l'aide du programme original et du fichier original de 700 000 blocs, nous avons créé 10 fichiers tests qui serviront pour faire les trois séries de tests présentés au prochain chapitre. L'exécution du programme paramninv.cpp permet cette opération. Pour la

réduction des fichiers, l'un ou l'autre des quatre programmes peut être utilisé. L'usager spécifie par la suite le nom du fichier (pour les tests, nous avons utilisé le fichier original de 700 000 blocs) ainsi que les dimensions des blocs. Puis le programme donne les coordonnées inférieures et supérieures du gisement ainsi que le nombre total de blocs à l'intérieur du gisement. Le programme offre par la suite la possibilité de réduire le gisement. Si cette option est choisie, le programme demande alors les bornes du nouveau gisement. Par la suite, le programme demande à l'opérateur s'il désire poursuivre l'optimisation de ce gisement immédiatement ou non. Si la réponse est "oui" alors le fichier sous forme standard de ce gisement ne sera pas créé et l'optimisation sera exécutée. À la fin, on obtient un fichier de sortie avec deux colonnes ajoutées au fichier sous forme standard, comme nous l'avons déjà mentionné dans la section précédente. Si la réponse est "non", alors le fichier sous forme standard de ce nouveau gisement sera créé et le programme demande le nouveau nom du fichier.

Après avoir observé le gisement initial de 700 000 blocs, nous nous sommes rendus compte qu'il y avait un très grand nombre de blocs stériles et que la zone minéralisée se trouvait exclusivement d'un seul côté du gisement. Nous avons donc choisi un point central au fond de la zone minéralisée et nous avons créé les fichiers tests à partir de ce point central. Les raisons pour lesquelles nous avons procédé de cette façon sont les suivantes: 1) on voulait éliminer le plus possible de blocs d'air en calculant nos niveaux du bas vers le haut, 2) on voulait avoir des gisements économiques de façon à ce que le flot maximum réussisse à extraire des blocs et finalement 3) on voulait que la répartition des teneurs soit sensiblement la même pour tous les gisements. Ce dernier point était important pour que la variation des teneurs ne devienne pas une variable lors des tests sur le nombre de blocs variables et le nombre de niveaux variables. Bien entendu, ces facteurs ne sont importants que pour les tests qui seront faits pour des gisements avec des valeurs de coûts réels.

Les 6 premiers fichiers créés ont tous environ 100 000 blocs, mais le nombre de niveaux varie de l'un à l'autre. On commence à 11 niveaux et on augmente de 5 niveaux chaque

fois de façon que le dernier fichier ait 36 niveaux. Les 5 autres fichiers ont tous le même nombre de niveaux, soit 14, mais cette fois-ci, c'est le nombre de blocs qui augmente graduellement par saut de 20 000 blocs. Ainsi on a un fichier de 20 000 blocs, un de 40 000 blocs, un de 60 000 blocs, un de 80 000 blocs et six de 100 000 blocs. Lorsqu'on parle du nombre de blocs, il s'agit du nombre de blocs compris dans le cône d'extraction tel que montré à la figure 5.3 de la section précédente. Les gisements rectangulaires en tant que tel contiennent beaucoup plus de blocs. Le tableau 5.2 et le tableau 5.3 qui suivent donnent les spécifications pour les 11 fichiers. La notation utilisée pour les fichiers va selon le nombre de blocs qu'ils contiennent dans chacune des trois dimensions. Ainsi le fichier 656614.txt est le fichier qui représente le gisement rectangulaire qui comprend 65 blocs selon l'axe des X, 66 blocs selon l'axe des Y et 14 niveaux. Si on multiplie chacune de ces valeurs ensemble, on obtient le nombre de blocs contenus dans le gisement rectangulaire, soit 60 060 blocs pour ce gisement, comme à la figure 5.2. Toutefois, sur ces 60 060 blocs il n'en restera qu'environ 40 000 lorsque le graphe des prédécesseurs aura été calculé, ce qui représente en fait le cône d'extraction et par le fait même, le graphe qui sera soumis au flot maximum. Les tableaux, comme nous pouvons le voir, donnent les informations suivantes : 1) le numéro du gisement, 2) le nom du fichier, 3) les nombres de blocs en direction des x, des y et des z, 4) le nombre de blocs à l'intérieur du gisement et 5) le nombre de blocs à l'intérieur du cône d'extraction. Pour le tableau 5.2 on a une information supplémentaire, à savoir l'écart du nombre de blocs de plus ou de moins que 100 000.

Tableau 5.2 : Spécifications des fichiers servant aux tests sur le nombre de niveaux

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué	diff. 100000
1	1398011	139	80	11	122320	99770	oui	230
2	1257116	125	71	16	142000	99920	oui	80
3	1306221	130	62	21	169260	100100	oui	-100
4	1385726	138	57	26	204516	99866	oui	134
5	1276031	127	60	31	236220	100130	non	-130
6	1046936	104	69	36	258336	100030	non	-30

Tableau 5.3 : Spécifications des fichiers servant aux tests sur le nombre de blocs

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué
7	505014	50	50	14	35000	20076	non
8	656614	65	66	14	60060	39494	oui
9	768014	76	80	14	85120	60004	oui
10	978014	97	80	14	108640	79702	oui
11	1357114	135	71	14	134190	99974	oui

CHAPITRE VI
RÉSULTATS DES TESTS SUR LES STRATÉGIES D'ACCÉLÉRATION DU FLOT MAXIMUM

À la lumière des résultats d'analyse de l'algorithme du flot maximum présentés au chapitre III, nous voulions effectuer trois types de tests qui confirmeraient les dires de Tachefine, de Hochbaum et Chen et les nôtres. Pour ce faire, nous avons effectué des tests qui nous ont permis de vérifier de quelle façon le flot maximum réagissait lorsqu'on faisait varier tour à tour le nombre de blocs, le nombre de niveaux et le pourcentage minerai/stérile. Pour isoler ces paramètres, nous avons créé différents fichiers à l'aide du programme présenté dans le chapitre précédent. Pour les sections suivantes, les résultats qui ont été obtenus pour chacun des paramètres seront présentés. Tous les résultats détaillés des tests sont présentés en annexe IV sous forme de tableaux.

6.1 Nombre de blocs

De façon à vérifier les dires de Tachefine, nous avons entrepris de faire une série de tests où la variable serait le nombre de blocs dans le gisement. Tachefine avait remarqué que le temps de résolution du flot maximum variait de façon linéaire en fonction du nombre de blocs. Les tests ont été effectués à l'aide des 5 derniers fichiers présentés dans le chapitre précédent, soit les fichiers 7, 8, 9, 10 et 11. Des tests ont été faits pour chacun des gisements avec des coûts réels et avec des coûts aléatoires. Pour les coûts réels, le flot maximum a été exécuté une seule fois étant donné que les coûts restent fixes. Pour les coûts aléatoires, le flot maximum a été exécuté 10 fois pour chacun des gisements de façon à obtenir des résultats plus uniformes et ainsi éviter d'obtenir des valeurs extrêmes qui pourraient biaiser les résultats. Ainsi, tous les paramètres, à l'exception du nombre de blocs, demeurent fixes, soit le nombre de niveaux à 14 et le pourcentage de stérile à 70% pour le calcul des coûts aléatoires seulement. En Annexe IV, les tableaux A4.7, A4.8, A4.10, A4.11 et A4.12 donnent tous les résultats des tests. La figure 6.1 présente

un graphique des moyennes de temps de calcul pour les 10 itérations lorsque le nombre de blocs varie et que les coûts sont attribués de façon aléatoire. La figure 6.2 présente le graphique du temps de calcul lorsque le nombre de blocs varie mais que les coûts sont calculés à l'aide des teneurs réelles de chacun des blocs et un prix du métal à 32$/tonne.

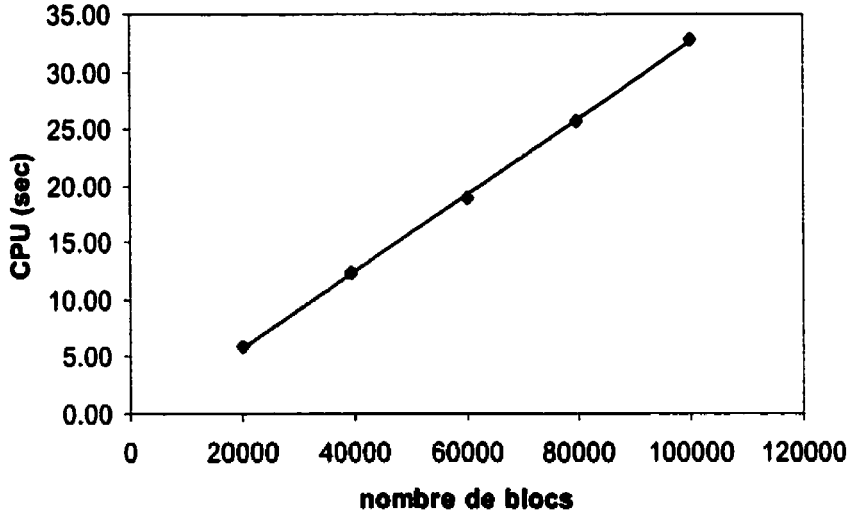


Figure 6.1 : CPU vs le nombre de blocs (génération de coûts aléatoire)

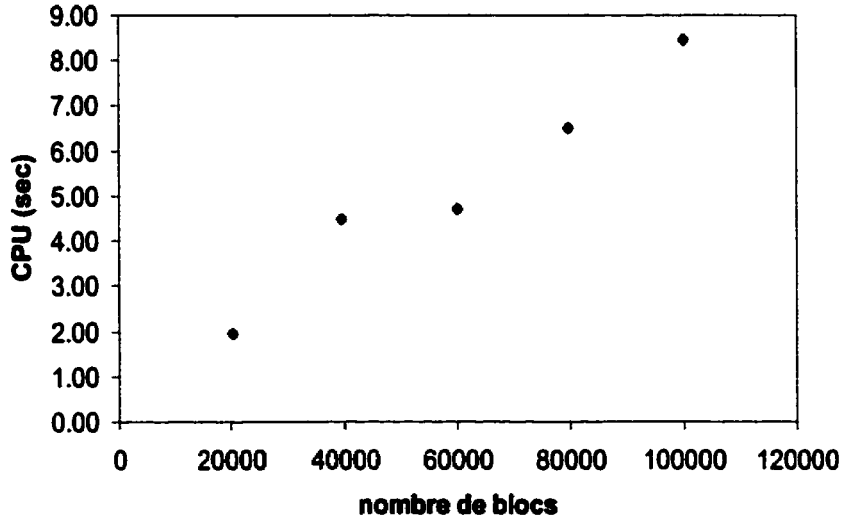


Figure 6.2 : CPU vs le nombre de blocs (pour un prix fixe de 32$)

Avec ces deux graphiques, on remarque un accroissement linéaire du temps de résolution selon le nombre de blocs. Lorsque les coûts sont générés de façon aléatoire, on obtient une courbe linéaire sans aucune déviation. Pour les calculs avec les coûts réels, on observe quelques petites déviations. Ces déviations peuvent être le résultat des variations de teneurs d'un gisement à l'autre. Ainsi, même si nous avons pris la peine de centrer tous nos gisements autour d'un même bloc, comme nous l'avons mentionné dans la section 4.3, les résultats montrent que, pour un gisement réel, les valeurs des teneurs varient suffisamment pour expliquer une telle déviation. Mais en somme, les résultats sont tout à fait conforment à ce que Tachefine avait observé.

Ainsi on remarque que la tendance des tests n'est pas conforme à ce que l'analyse de complexité laissait croire dans le chapitre IV. Si on retourne à ce chapitre, l'analyse de complexité de l'algorithme du *Highest-label preflow-push* révèle que les temps de calcul auraient dû varier selon $O(n^2m^{1/2})$. Le fait que l'algorithme varie de façon linéaire en fonction du nombre de blocs, pour le cas particulier du problème des contours ultimes dans une mine à ciel ouvert, est extrêmement intéressant. Cela signifie que le temps de résolution augmente pratiquement de façon linéaire avec la taille des problèmes. Ainsi, nous sommes loin du cas extrême.

6.2 Nombre de niveaux

Lors de ses tests, Tachefine avait fait varier le nombre de blocs sans toutefois tenir compte du nombre de niveaux. À la suite à l'analyse de l'algorithme de flot maximum qui a été faite au chapitre IV, nous croyons que le nombre de niveaux pourrait avoir une certaine incidence sur le temps de calcul de l'algorithme puisque ce dernier influence la longueur des chemins du nœud s au nœud t. C'est pourquoi nous avons entrepris une série de tests où la variable serait le nombre de niveaux. Les tests ont été effectués à l'aide des 6 premiers fichiers présentés dans le chapitre précédent, soit les fichiers 1, 2, 3, 4, 5 et 6. Comme pour les tests avec le nombre de blocs variable, des tests ont été réalisés pour chacun des gisements avec des coûts réels et avec des coûts aléatoires.

Pour les coûts réels, le flot maximum a été lancé une seule fois étant donné que les coûts restent fixes. Pour les coûts aléatoires, le flot maximum a été lancé 10 fois pour chacun des gisements. Ainsi, tous les paramètres, à l'exception du nombre de niveaux, ont été fixés, soit le nombre de blocs à 100 000 et le pourcentage de stérile à 70% pour le calcul des coûts aléatoires seulement. En Annexe IV, les tableaux A4.1, A4.2, A4.3, A4.4, A4.5 et A4.6 donnent tous les résultats des tests. La figure 6.3 présente un graphique des moyennes de temps de calcul pour les 10 itérations lorsque le nombre de niveaux varie et que les coûts sont attribués de façon aléatoire. La figure 6.4, quant à elle, présente le graphique du temps de calcul lorsque le nombre de niveaux varie mais que les coûts sont calculés à l'aide des teneurs réelles de chacun des blocs et d'un prix du métal à 32$/tonne.

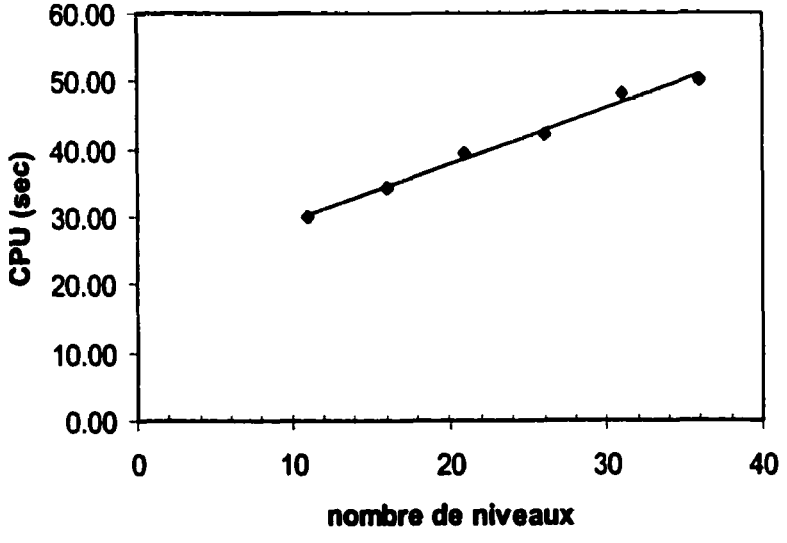


Figure 6.3 : CPU vs nombre de niveaux (génération de coûts aléatoires)

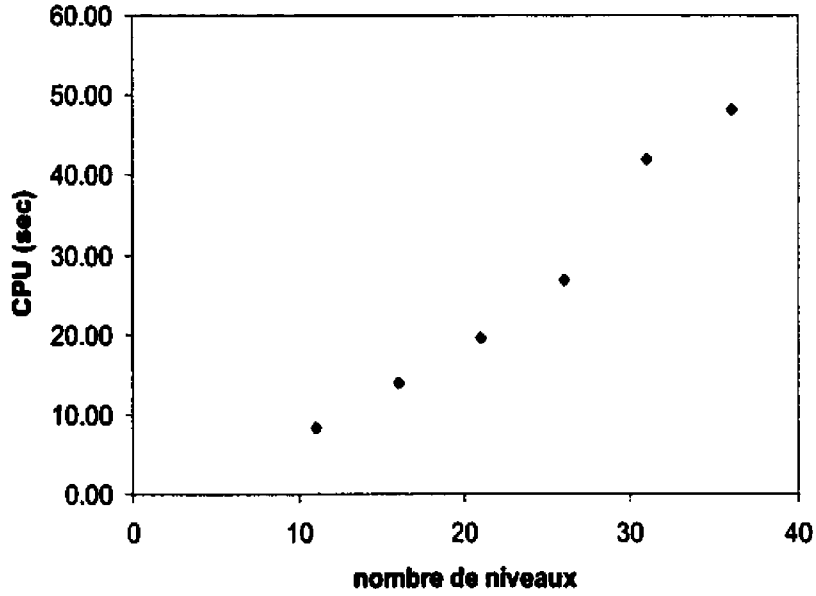


Figure 6.4 : CPU vs nombre de niveaux (pour un prix fixe de 32$)

Les résultats indiquent que le temps de résolution croît de façon linéaire avec le nombre de niveaux. Encore une fois, on remarque que les points du graphique qui représentent les coûts réels ne se retrouvent pas tout à fait sur la ligne.

À cette étape, il serait difficile de dire lequel des deux paramètres, le nombre de blocs ou le nombre de niveaux, est le plus déterminant.

6.3 Graphe inversé et % minerai-stérile

Finalement, la troisième série de tests servira à vérifier les dires de Hochbaum et Chen. Cette fois-ci, nous vérifierons l'impact d'inverser le graphe lorsque le pourcentage minerai/stérile varie. Nous tenterons de déterminer ces circonstances qui permettent d'accélérer l'exécution des calculs pour l'algorithme du flot maximum. Les tests ont été effectués à l'aide d'un seul fichier à partir duquel nous n'avons fait varier que le pourcentage minerai/stérile sans que le graphe ne soit inversé. Puis nous avons refait les même tests pour différents ratio minerai/stérile, mais cette fois-ci, sur un graphe inversé.

Pour tous les tests, les coûts ont bien entendu été déterminés de façon aléatoire, de manière à obtenir le ratio minerai/stérile voulu. Ainsi, pour chaque ratio évalué, le flot maximum a été exécuté 10 fois de façon à obtenir des résultats plus uniformes. Étant donné la grande quantité d'itérations qui devaient être faites, le fichier utilisé pour ces calculs est le fichier 8, soit celui qui compte 40 000 blocs et 14 niveaux. Nous avons choisi ce fichier parce que l'exécution était quand même assez rapide, mais pas assez pour qu'on ne puisse pas voir une différence dans les temps de résolution. Ainsi, tous les paramètres, à l'exception du pourcentage de stérile, ont été fixés, soit le nombre de niveaux à 14 et le nombre de blocs à 40 000. En Annexe IV, les tableaux A4.8 et A4.9 donnent tous les résultats des tests. Les figures qui suivent résument ces tableaux qu'on retrouve en annexe. La figure 6.5 représente les résultats des tests de la variation du pourcentage du nombre de blocs de stérile.

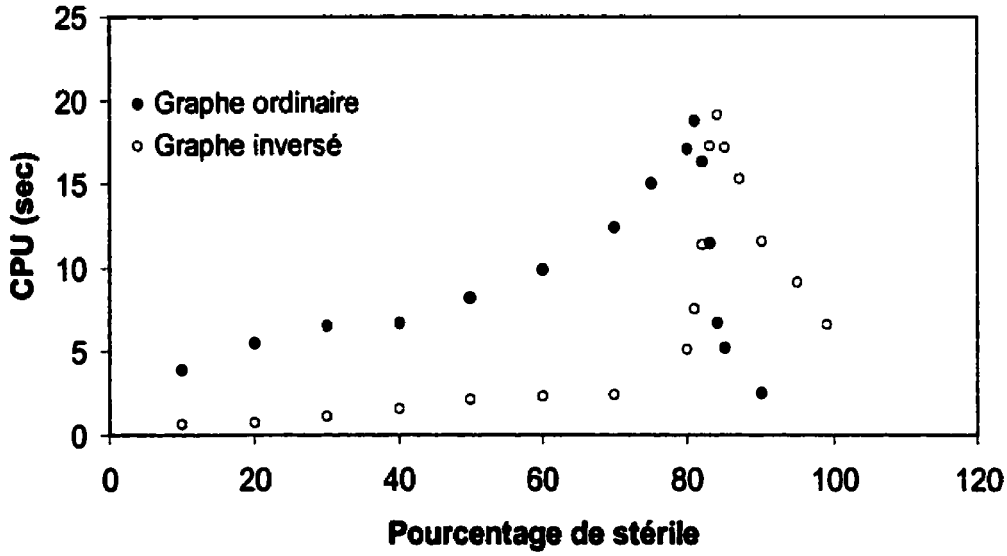


Figure 6.5 : Graphique de la moyenne des temps de résolution en fonction du pourcentage de stérile

On remarque que le graphe inversé produit des résultats beaucoup plus rapides jusqu'à un ratio de stérile d'environ 82%. Par la suite, c'est le graphe initial qui produit les

résultats les plus rapides. De plus, on remarque que les deux courbes sont symétriques par rapport à une ligne qui se situerait aux alentours de 82%. Ainsi, on remarque que le maximum de temps de calcul est atteint juste avant la ligne de symétrie. Mais que représente exactement ce chiffre de 82% de stérile? Serait-il le même si nous changions les valeurs des blocs?

Pour expliquer ce qui se produit réellement, nous nous référerons à la figure 6.6 qui explique la différence entre le graphe inversé et le graphe ordinaire.

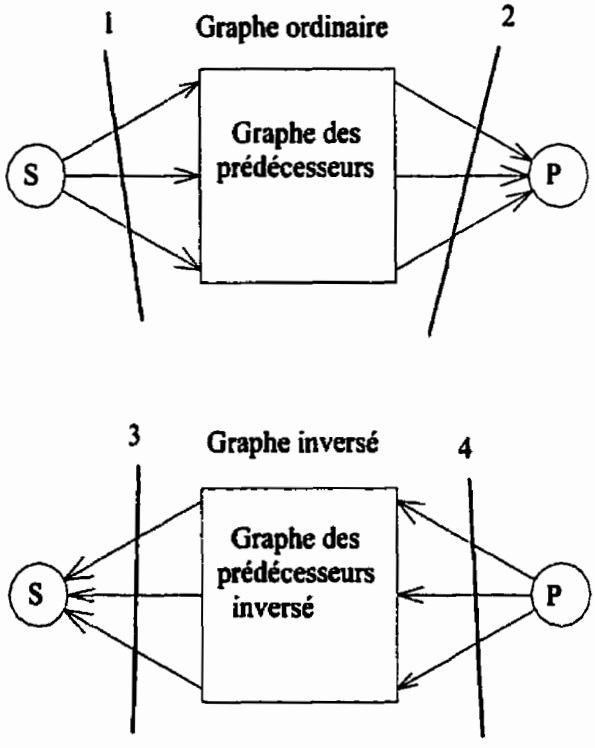


Figure 6.6 : Explications sur la nature des résultats obtenus

Les lignes qui portent les numéros 1 et 3, sur la figure, représentent les coupes au nœud source. Les arcs qui traversent la coupe 1 pointent vers tous les blocs de minerai du gisement, tandis que ceux qui traversent la coupe 3 arrivent de tous les blocs de minerai. Les lignes qui portent les numéros 2 et 4 sur la figure représentent les coupes au nœud

puits. Les arcs qui traversent la coupe 2 arrivent de tous les blocs de stérile tandis que ceux qui traversent la coupe 4 pointent vers tous les blocs de stérile du gisement. Ainsi :

Coupe $_1$ = Coupe $_3$ = Σvaleurs des blocs de minerai, et

Coupe $_2$ = Coupe $_4$ = Σvaleurs des blocs de stérile

En sachant cela, nous avons fait un deuxième graphique à l'aide des mêmes résultats, mais cette fois qui donne le temps d'exécution en fonction du ratio de la valeur de la coupe au nœud source sur la valeur de la coupe au nœud puits. Ainsi, si:

1. $\frac{coupe\_source}{coupe\_puit} < 1 \Rightarrow \sum$ valeur des blocs de stérile $> \sum$ valeur des blocs de minerai

2. $\frac{coupe\_source}{coupe\_puit} = 1 \Rightarrow \sum$ valeur des blocs de stérile $= \sum$ valeur des blocs de minerai

3. $\frac{coupe\_source}{coupe\_puit} > 1 \Rightarrow \sum$ valeur des blocs de stérile $< \sum$ valeur des blocs de minerai

En utilisant une échelle logarithmique pour les valeurs du ratio, on se rend compte qu'on obtient une allure de courbe qui ressemble étrangement à la première.

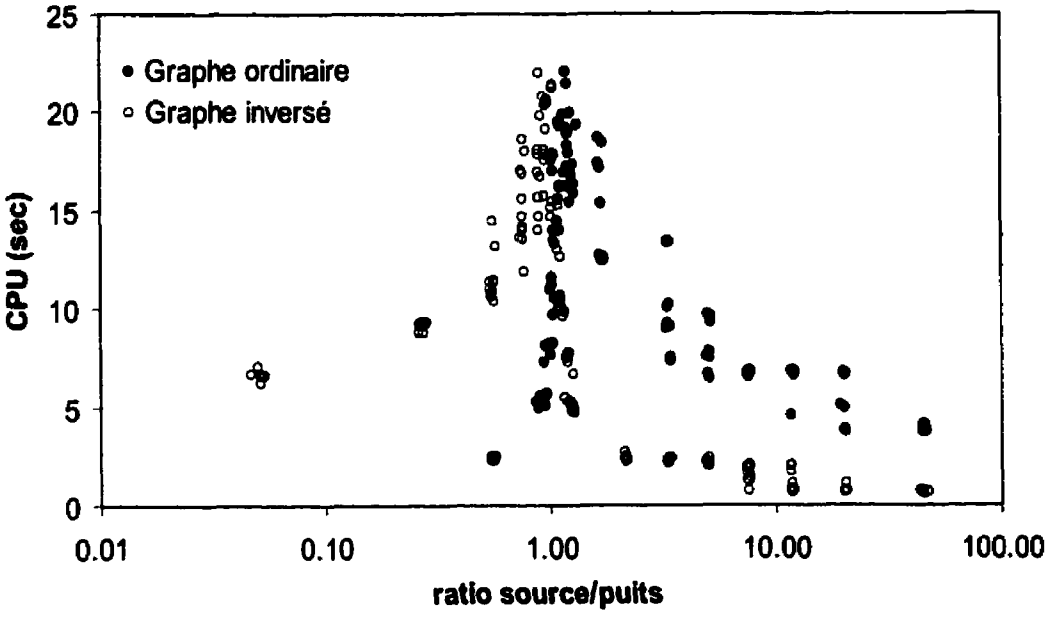


Figure 6.7 : Graphique du temps de résolution en fonction du ratio de la valeur de la coupe au nœud source sur la valeur de la coupe au nœud puits

Ainsi, à l'aide de cette figure on en déduit que si:

1. $\frac{coupe\_source}{coupe\_puit} < 1 \Rightarrow$ il est préférable d'utiliser un graphe ordinaire,

2. $\frac{coupe\_source}{coupe\_puit} = 1 \Rightarrow$ on peut utiliser l'un ou l'autre des graphes, et

3. $\frac{coupe\_source}{coupe\_puit} > 1 \Rightarrow$ il est préférable d'utiliser le graphe inversé.

Il est bon de spécifier que la majorité des cas réels ressembleront surtout à la troisième option, c'est-à-dire que généralement, pour qu'un gisement soit exploité, il faut que la valeur des blocs de minerai soit supérieure à la valeur des blocs de stérile.

De cette façon, on peut déduire que le sommet qui se situe à une valeur d'environ 82% qu'on observe sur la figure 6.5, représente en fait le point où le ratio est égal à 1 pour le gisement et les coûts que nous avons utilisés sur la dernière figure. Ainsi, ce n'est pas le nombre de blocs de minerai ou de stérile qui influence la rapidité d'exécution mais bien les capacités des coupes.

Mais pourquoi la courbe de la figure 6.7 a-t-elle cette allure? Autrement dit, pourquoi atteint-on un maximum puis redescend-on par la suite? Pour tenter d'expliquer ce phénomène, on se référera à la façon dont l'algorithme du «highest push-relabel» pousse le flot le long du graphe. En partant de la source, il commence, lors de l'étape du pré-traitement, par saturer tous les arcs (s, j). Par la suite, il essaie de distribuer le flot en excès de façon à atteindre le puits le plus rapidement possible.

Supposons qu'il y a beaucoup plus de minerai que de stérile, c'est à dire que le ratio de la valeur de la coupe au nœud source sur la valeur de la coupe au nœud puits est plus grand que 1. Alors l'algorithme sature rapidement tous les arcs, mais il continue à chercher des chemins pour les neouds actifs encore présents dans sa liste. De plus, il doit quand même ramener le flot excédentaire vers la source.

Supposons maintenant qu'il y a une quantité presque égale de minerai et de stérile, c'est à dire que le ratio de la valeur de la coupe au nœud source sur la valeur de la coupe au nœud puits est presque égal à 1. Alors le nombre de chemins possibles est beaucoup plus grand et l'algorithme doit travailler plus fort pour passer toutes ses unités de flots de la source vers le nœud puits. De plus, si tout le flot ne peut traverser, alors il doit en ramener une partie. Ce qui pourrait expliquer que l'algorithme soit beaucoup plus lent juste avant que la valeur du ratio n'atteigne 1.

Finalement, supposons qu'il y a beaucoup moins de minerai que de stérile, c'est à dire que le ratio de la valeur de la coupe au nœud source sur la valeur de la coupe au nœud puits est plus petit que 1. Alors les arcs saturés qui sortent de la source trouvent facilement preneurs et tout le flot envoyé lors de l'étape de pré-traitement est dirigé vers

le nœud puits, la liste des nœuds actifs est rapidement épuisée et ce, sans qu'il ne soit nécessaire de ramener du flot vers le nœud source.

Toutes ces suppositions pourraient expliquer pourquoi l'algorithme est beaucoup plus lent autour d'un ratio égal à 1 et pourquoi il est un peu plus lent lorsque le ratio est plus grand que 1 que lorsqu'il est plus petit que 1.

Ainsi les tests qui ont été faits et qui utilisent cette dernière stratégie confirment les résultats préalablement obtenus par Hochbaum et Chen. Selon nous, il s'agit là de l'élément déterminant pour la rapidité du flot maximum. Toutefois, le nombre de blocs peut devenir une variable si on a une certaine flexibilité sur l'ampleur des dynamitages. Ce facteur est lié à la taille des équipements utilisés. La grosseur de blocs dépend donc du moment où s'effectue l'optimisation, c'est-à-dire avant ou après la mise en exploitation de la mine, et si on a de la flexibilité sur le choix des équipements. De plus, la distribution statistique des teneurs de même que la distribution spatiale pourraient avoir une certaine influence sur la rapidité d'exécution de l'algorithme. Ainsi, si tout le métal se retrouve dans un seul bloc et que l'on a un rapport de 1, si le bloc est en surface, le temps de calcul sera instantané. Par contre, s'il est en profondeur, l'algorithme devra saturer tous les arcs allant au puits avant de terminer. De la même façon, des teneurs plus regroupées dans l'espace devraient mener à des temps d'exécution plus rapides.

CHAPITRE VII
EXTENSION ET CONCLUSION

Lorsqu'on utilise un programme de planification minière tel que celui qui est présenté dans ce mémoire, il est important de bien garder en mémoire que l'exactitude des résultats obtenus dépend d'abord et avant tout de la précision avec laquelle le modèle de blocs a été fait. Si les données initiales sont erronées, les résultats finaux le seront aussi. Ainsi la confiance qu'on aura dans les résultats qui nous seront donnés dépend de la confiance qu'on a dans les données initiales.

À la suite de la recherche bibliographique qui a été faite au chapitre III, la méthode choisie qui nous semble avoir le plus de potentiel pour résoudre les problèmes de planification dans les mines à ciel ouvert est celle qui utilise l'algorithme du flot maximum. Cette méthode permet non seulement de calculer les contours optimaux d'une mine à ciel ouvert, mais peut aussi être utilisée pour calculer les séquences de production lorsqu'elle est utilisée comme sous-routine lors d'une relaxation lagrangienne. Bien entendu, étant donné la nature circulaire du problème, présentée au chapitre III, il ne faut pas croire qu'il s'agisse là d'une méthode exacte, mais c'est, à notre avis, celle qui s'en approche le plus et qui a le plus de potentiel pour y parvenir.

Comme nous l'avons vu dans le chapitre III, plusieurs auteurs se sont basés sur l'algorithme de Lerchs et Grossmann lors de leur recherche de méthodes efficaces pour calculer les contours ultimes d'une mine à ciel ouvert. Or, la majorité de ces méthodes développées au cours des ans utilisent des réductions de graphe qui permettent d'améliorer les temps de calcul de façon considérable. Lorsqu'on a commencé à utiliser des flots maximum pour résoudre les mêmes types de problèmes, on a continué à utiliser de telles techniques de réduction de graphe sans pour autant s'interroger sur la pertinence de la chose. Comme il a été prouvé au chapitre IV, il est inutile d'utiliser de telles

techniques de pré-traitement parce que l'algorithme du flot maximum effectue implicitement ce genre de réduction durant l'opération d'étiquetage des nœuds.

De plus, dans le chapitre IV, nous démontrons que l'algorithme du flot maximum appliqué au problème des contours ultimes ne réagit pas comme on pourrait s'y attendre, compte tenu de l'analyse de la complexité des algorithmes utilisés. Afin de prouver ces dires, des tests ont été effectués à l'aide de programmes que nous avons créés et des fichiers qui nous ont été fournis par la mine «M». Les résultats des tests, présentés dans le chapitre V, montrent clairement que le temps de résolution croît de façon linéaire avec le nombre de blocs ou le nombre de niveaux dans la mine. Ces résultats sont fort importants, car ils supposent que cette méthode de résolution peut être envisagée pour des problèmes de plus grande taille sans que le temps de résolution n'augmente de façon démesurée. De plus, en faisant varier le pourcentage minerai stérile, on remarque que la meilleure amélioration possible pour accélérer le temps de résolution du flot maximum est d'inverser le graphe lorsque le ratio de la valeur de la coupe au nœud source sur la valeur de la coupe au nœud puits est supérieur à 1.

Sur la base de ce mémoire, les travaux de recherche futurs pourraient d'abord tenter d'insérer différents modules omis lors de la programmation initiale, c'est-à-dire d'améliorer le programme de façon à pouvoir considérer entre autres les pentes variables, les routes de halage, plusieurs types de minerai et calculer les coûts de façon à représenter plus fidèlement la réalité. De plus, il faudra régler quelques lacunes de programmation qui concernent majoritairement l'algorithme du flot maximum. Sur le plan pratique, il faudrait en faire une version plus flexible qui puisse fonctionner autant sur une plate-forme de travail UNIX ou LINUX que sur Windows. De plus, il faut en améliorer le code de façon à traiter des problèmes plus grands (de l'ordre de 800 000 blocs). De plus, le programme de flot maximum actuel ramène le flot automatiquement. Il serait très intéressant de le modifier de façon à ne garder que la coupe minimale étant donné que c'est vraiment ce qui nous intéresse. Cette modification pourrait améliorer un peu la rapidité d'exécution des calculs du flot maximum. Éventuellement, il serait aussi

intéressant d'optimiser le code de tout le programme pour le rendre plus puissant. Une fois ces ajustements réalisés, on pourra penser à inclure le calcul des séquences de production selon la méthode de Dagdeleen telle que présentée par Tachefine.

Éventuellement, il serait très intéressant de comparer les stratégies du *HL* et du *LIFO* combiné à un algorithme de «gap-relabeling» pour voir lequel des deux s'avérerait le plus prometteur, ces tests n'ayant pas été faits auparavant.

Il aurait été également très intéressant de comparer nos temps de calcul avec ceux que nous aurions pu obtenir à l'aide du logiciel Whittle. Initialement prévus, ces tests n'ont pas pu se faire car ils auraient nécessité la simplification du calcul des coûts fait par Whittle pour obtenir l'équivalent de ce que nous faisions. Cette simplification était très ardue compte tenu de la lourdeur du calcul des coûts avec le logiciel Whittle et n'aurait pas permis une comparaison adéquate des deux approches.

En conclusion, nous croyons avoir produit un outil facile d'utilisation, qui se greffe facilement à un logiciel commercial de planification minière et qui pourra, avec quelques améliorations et raffinements, produire des solutions très proches de l'optimum en regard des problèmes de planification dans les mines à ciel ouvert et ce, dans un temps relativement court comparé à ce qui est fait présentement dans la pratique.

CHAPITRE VIII
RÉFÉRENCES

AHUJA, R.K., MAGNANTI, T.L. et ORLIN, J.B. (1993). Network flows : theory, algorithms, and applications, Prentice-Hall, inc. Upper Saddle River, New Jersey, USA. 846 p.

AKAINE, A. et DAGDELEN, K. (1999). A strategic production scheduling method for an open pit mine. Proc., 28th APCOM, 729-738.

ALFORD, C.G. et WHITTLE, J. (1986). Application of Lerchs-Grossman pit optimization to the design of open-pit mines. The AusIMM/IE Aust Newman Combined Group, Large Open Pit Mining Conference, Australia, 201-207.

http://www.whittle.com.au

ARMSTRONG, D. (1990). Definition of mining parameters. Surface Mining, 2nd ed., Society for mining, metallurgy and exploration, U.S.A., 459-464.

BAAFI, E.Y., MILAWARMA, E. et CUSACK, C. (1995). Pit optimization of Sapan Dalam multi-seam coal deposit of Indonesia. Mine Planning and Equipment Selection, Singhal et al. (eds), Balkema, Rotterdam, ISBN 90 5410 5690, 3-8.

BAAFI, E.Y., MILAWARMA, E. et CUSACK, C. (1995). Computer aided design and analysis of Sapan Dalam open pit mine. Proc., 25th APCOM, 283-288.

BARNES, R.J. et JOHNSON, T.B. (1982). Bounding techniques for the ultimate pit limit problem. Proc. 17th APCOM, 263-273.

BRATICEVIC, D.B. (1984). Open-pit optimization method. Proc. 18th APCOM, 133-137.

BERLANGA, J.M., CARDONA, R. et IBARRA, M.A. (1989). Recursive formulae for the floating cone algorithm. International Journal of Surface Mining, 3, No 3, 141-150.

BLACKWELL, G.H. (1993). Computerized mine planning for medium-size open-pits. Trans. (Section A: Mining Industry), 102, A83 - A88.

BOHNET, E.L. (1990). Optimum production scheduling. Surface Mining, 2nd ed., Society for Mining, Metallurgy and Exploration inc., USA, 476-479.

CACETTA, L. et GIANNINI, L.M. (1985). On bounding techniques for the optimum pit limit problem. Proc. Aust. Inst. Min. Metall., 290, No 4, 87-89.

CACETTA, L. et GIANNINI, L.M. (1986). Optimization techniques for the open pit limit problem. Proc. Aust. Inst. Min. Metall., 291, No 8, 57-63.

CACETTA, L. et GIANNINI, L.M. (1988). The generation of minimum search pattern in the optimum design of open pit mines. Proc. Aust. Inst. Min. Metall., 293, No 5, 57-61.

CAI, W.L. (1992). Sensitivity analysis of 3-D model block dimensions in the economic open pit limit design. Proc. 23rd APCOM, (Littleton, Colorado : AIME), 475-486.

CAI, W.L. et BANFIELD, A.F. (1993). Long range open pit sequencing – A comprehensive approach. Proc. 24th APCOM, 2, 11-18.

CHEN, T. (1977). 3D pit design with variable wall slope capabilities. Proc. 14th APCOM. AIME, New – York, 615-625.

COLEOU, T. (1985). MULTIPIT program. User's manual and case study. Centre de Géostatistique, Fontainebleau, France.

COLEOU, T. (1987). Le paramétrage technique des réserves. Science de la Terre.

COLEOU, T. (1987). Paramétrage technique des réserves et optimisation d'un projet minier. Thèse de Doctorat, Centre de Géostatistique, Fontainebleau, France.

COLEOU, T. (1988). L'optimisation de carrières: méthodes, algorithmes et mise en œuvre informatique dans le programme MULTIPIT. Centre de Géostatistique, Fontainebleau, France.

COLEOU, T. (1988). Technical parameterization of reserves for open pit design and mine planning. Proc. 21st APCOM, 485-494.

COLLINS, J.L. (1994). Converting an optimum pit based on block modelling into a practical pit based on string modelling. International Journal of Surface Mining, Reclamation and Environment, 8, No 4, 167-169.

CRAWFORD, J.T. (1979). Open pit limit analysis - some observations on its use. Proc. 16th APCOM, (New York: AIME), 625-634.

CRAWFORD, J.T. et DAVEY, R.K. (1979). Case study in open-pit limit analysis. Computer Methods for the 80's in the Mineral Industry, AIME, New-York, 310-317.

DAGDELEN, K. et JOHNSON, T.B. (1986). Optimum open pit mine production scheduling by lagrangian parametrization. Proc. 19th APCOM, 127-141.

DANTZIG, G.B. et WOLF, P. (1961). The decomposition method for linear programming. Econometrica, 29, 767-778.

DAVID, M., DOWD, P. et KOROBOV, S. (1974). Forecasting departure from planning in open pit design and grade control. Proc. 12th APCOM, 2, F131-F142.

DAVIS, R.E. et WILLIAMS, C.E. (1973). Optimization procedures for open pit mine scheduling. Proc. 11th APCOM, 1C, University of Arizona, Tucson, USA, C1-C18.

DENBY, B. et SCHOFIELD, D. (1994). Open-pit design and scheduling by use of genetic algorithms. Trans (Section A: Mining Industry), IMM, 103, A21 - A26.

DENBY, B. et SCHOFIELD, D. (1995). Inclusion of risk assesment in open-pit design and scheduling. Trans (Section A: Mining Industry), IMM, 104, A67 - A71.

DENBY, B., SCHOFIELD, D. et HUNTER, G. (1996). Genetic algorithms for open pit scheduling – Extension into 3 dimensions. Mine Planning and Equipment Selection, Hennies, Ayres da Silva et Chaves (eds), Balkema, Rotterdam, ISBN 90 5410 8274 4, 177-186.

DIERING, J.A.C. (1984). Algorithm for variable slope pit generation. Proc. 18th APCOM, 113-121.

DOWD, P.A. et ONUR, A.H. (1992). Optimising open pit design and sequencing. Proc. 23rd APCOM, (Littleton, Colorado : AIME), 411-422.

DOWD, P.A. and ONUR, A.H. (1993). Open-pit optimization - part 1 : optimal open-pit design. Trans (Section A: Mining Industry), IMM, 102, A95 - A104.

ELEVLI, B. (1995). Open pit mine design and extraction sequencing by use of OR and AI concepts. International Journal of Surface Mining, Reclamation and Environment, 9, No 4, 149-153.

FORD, L.R. et FULKERSON, D.R. (1956). Maximal flow through a network. Canadian Journal of Mathematics, 8, 399-404.

FRANÇOIS-BONGARÇON, D. (1978). Le paramétrage des contours optimaux d'une exploitation à ciel ouvert. Thèse PhD, Institut National Polytechnique de Lorraine, Centre de géostatistique, Fontainebleau, France, avril 1978.

FRANÇOIS-BONGARÇON, D. et GUIBAL, D. (1980). Open pit optimization by parametrization of the final pit contour. New advances in applications. Proc. 17th APCOM, Moscow.

FRANÇOIS-BONGARÇON, D. et GUIBAL, D. (1984). Parameterization of optimal designs of an open pit- beginning of a new phase of research. Trans. SME. AIME, 274, 1801-1805.

FRANÇOIS-BONGARÇON, D. et MARÉCHAL, A. (1976). Algorithme d'optimisation de carrière finale par la méthode de paramétrage de contours optimaux. Confidential Note, Centre de Géostatistique, Fontainebleau, France.

FRANÇOIS-BONGARÇON, D. et MARÉCHAL, A. (1977). A new method for oen-pit design: Parameterization of the final pit contour. Proc. 14th APCOM, New-York, 573-583.

FRIMPONG, S. et ACHIREKO, P.K. (1997). The MCS/MFNN algorithm for open pit optimization. International Journal of Surface Mining, Reclamation and Environment, 11, No 1, 45-52.

FYTAS, K., PELLEY, C. et CALDER, P. (1987). Optimization of open pit short and long range production scheduling. CIM Bulletin, 80, No 904, 55-61.

FYTAS, K., HADJIGEORGIOU, J. et COLLIN, J.L. (1993). Production scheduling optimization in open pit mines. International Journal of Surface Mining and Reclamation, 7, No 1, 1-9.

GANGWAR, A. (1982). Using geostatistical ore block variances in production planning by integer programming. . Proc. 17th APCOM, Colorado School of mines, 443-460.

GAUTHIER, F.J. et GRAY, R.G. (1971). Pit design by computer at Gaspe Copper Mines, Limited. CIM Bulletin, 64, No 11, 95-102.

GERSHON, M. E. (1982). A linear programming approach to mine scheduling optimization. Proc. 17th APCOM, Colorado School of mines, Golden, USA, 483-493.

GERSHON, M. E. (1987). An open-pit production scheduler : Algorithm and implementation. Minning Engineering, 39, No 8, August, 793-796.

GERSHON, M. E. (1987). Heuristic approaches for mine planning and production scheduling. International Journal of Mining and Geological Engineering, 5, 1-13.

GIANNINI, L.M., CACCETTA, L., KELSEY, P. and CARRAS, S. (1991). PITOPTIM: A new high speed network flow technique for optimum pit design facilitating rapid sensitivity analysis. Proc. of AusIMM, 296, No 2, 57-62.

GOLDBERG, A.V. and TARJAN, R.E. (1986). A new approach to the maximum flow problem. Proceedings of the 18th ACM Symposium on the Theory of computing, 136-146. Full paper in Journal of ACM, 35(1988), 921-940.

GORDON, S. T. (1996). Pit optimisation and mine production scheduling - the way ahead. Proc. 26th APCOM, 221-228.

HOCHBAUM, D.S. et CHEN, A. (1998). Performance analysis and best implementations of old and new algorithms for the open-pit mining problem. To appear in Naval Research Quartly. 40p.

http://riot.ieor.berkeley.edu/~dorit/pub/mine.html

HUTTAGOSOL, P. et CAMERON, R.E. (1989). Modified-tree graph algorithms for ultimate pit limit analysis. SME Annual Meeting, Las Vegas, Preprint no 89-24.

HUTTAGOSOL, P. et CAMERON, R.E. (1992). A computer design of ultimate pit limit by using transportation algorithm. Proc. 23rd APCOM, 443-460.

JIANG, Y.D. (1995). A modified network flow algorithm for pit limit optimization. 3rd CAMI, Montreal, Canada, Octobre 22-25, 139-148.

JOHNSON, T.B. (1968). Optimum open-pit mine production scheduling. PhD Dissertation, Dept. of IEOR, University of California, Berkeley, 120 p.

JOHNSON, T.B. (1969). Optimum open-pit mine production scheduling. A Decade of Digital Computing in the Mineral Industry, Weiss A. ed., (New York: AIME), 539-562.

JOHNSON, T.B. et BARNES, R.J. (1988). Application of the max flow algorithm to ultimate pit design. Engineering Design: Better results through operations research methods, Levary R.R. ed., 518-531.

JOHNSON, T.B. et MICKLE, D.G. (1970). Optimum design of an open-pit – An application in uranium. Proc. 9th APCOM, CIM Special Volume 12, Decision Making in the mineral Industry, Montréal, 331-338.

JOHNSON, T.B. et SHARP, W.R. (1971). A three dimensional dynamic programming method for optimal open-pit design. Rep. Invest. USBM 7553, 25p.

KIM, Y.C. (1978). Ultimate pit limit design methodologies using conputer models – The state of the art. Mining Engineering, 30, october, 1454-1459.

KIM, Y.C. (1979). Open–pit limits analysis, technical overview. Computer Methods for the 80's in the Mineral Industry, AIME, New-York, 297-303.

KIM, Y.C. et ZHAO, Y. (1994). Optimum open pit production sequencing – The current state of the art. SME Annual Meeting, Albuquerque, New Mexico, Preprint No 94-224, Frebruary 14-17.

KOENIGSBERG, E. (1982). The optimum contours of an open pit mine: An application of dynamic programming. Proc. 17th APCOM, 274-287.

KOROBOV, S. (1974). Method for determining optimal ultimate open pit limits. Department of Mineral Engineering, École Polytechnique de Montréal, Canada, Paper EP 74- R- 4.

KOROBOV, S.D. (1993). Parametric analysis of open pit mines. Proc. 24th APCOM, 57-66.

LEIGH, R.W. et BLAKE, R.L. (1970). An iterative approach to the optimal design of open pits. Proc. 9th APCOM, CIM Special Volume 12, Montreal, Canada, 254-260.

LEMIEUX, M. (1977). A different method of modelling a mineral deposit for a three-dimensional open pit computer design application. Proc. 14th APCOM, AIME, New-York, 557-572.

LEMIEUX, M. (1979). Moving cone optimizing algorithm. Computer Methods for the 80's in the mineral Industry, Weiss A. ed., (New-York: AIME), 329-345.

LERCHS, H. and GROSSMANN, I.F. (1965). Optimum design of open-pit mines. Trans. CIM Bulletin, 58, 47-54.

LIPKEWICH, M.P. et BORGMAN, L. (1969). Two- and three-dimensional pit design optimization techniques. A Decade of Digital Computing in the Mineral Industry, Weiss A. ed., AIME, New York, 505-523.

LIZOTTE, Y. (1988). The economics of computerized open-pit design. International Journal of Surface Mining, 2 , 59-78.

MARINO, J.M. et SLAMA, J.P. (1972). Ore reserve evaluation and open pit planning. Proc. 10th APCOM, (Johannesburg, South Africa: SAIMM), 139-144.

MASTORIS, J. et TOPUZ, E. (1995). Modeling, optimization and sensitivity analysis of the final pit limits for a lignite deposit. Mining Engineering, 47, No 11, 1027-1032.

MATHERON, G. (1975). Le paramétrage des contours optimaux. Note Géostatistique no. 128 Centre de Géostatistique et de Morphologie Mathématique, Internal report N-403, Fontainebleau, France, 54 p.

MATHERON, G. (1975b). Complément sur le paramétrage des contours optimaux. Note Géostatistique no. 129 Centre de Géostatistique et de Morphologie Mathématique, Internal report N-401, Fontainebleau, France, 54 p.

MATHERON, G. (1975c). Le paramétrage technique des réserves. Note Géostatistique no. 134 Centre de Géostatistique et de Morphologie Mathématique, Internal report N-453, Fontainebleau, France, 54 p.

MATHERON, G. (1975d). The transfer functions and their estimation. Advanced Geostatistics in the Mining Industry, NATO Advanced Study Institute Series, D. Reidel Publishing Co., Dordrecht, Holland.

MEYER, M. (1969). Applying linear programming to the design of ultimate pit limits. Management Science, 16, no 2, B 121-135.

ODINS, P. et HANSON, N. (1998). Don't close your pit, optimise it! 32nd Newcastle Symposium on Advances in the Study of Sydney Basin, Newcastle, Australia, april 98.
http://www.whittle.com.au

ONUR, A.H. et DOWD, P.A. (1993). Open pit optimization - part 2 : production scheduling and inclusion of roadways. Trans (Section A: Mining Industry), IMM, 102, A105 - A113.

PANA, M.T. (1965). The simulation approach to open–pit design, Proc. 5th APCOM, 2, Tucson, Arizona, ZZ1-24.

PHILLIPS, D.A. (1973). Optimum design of an open pit. Proc. 10th APCOM, 145-147.

PICARD, J.C. (1974). Maximum closure of a graph. Departement of Industrial Engineering, École Polytechnque de Montréal, Canada, paper EP 74-R-2.

PICARD, J.C. (1976). Maximum closure of a graph and applictions to combinatorial problems. Management Science, 22, No 11, 1268-1272.

PICARD, J.C. et SMITH, B.T. (1993). Optimal rate of return in open pit mine design. École Polytechnique de Montréal, Centre de recherche sur les transports - Publication #993, 22p.

PLASSE, C. et ELBROND, J. (1988). Les contours optimaux d'exploitations à ciel ouvert par un algorithme de fot avec SAS/OR. Computer Applications in the Mineral Industry, CAMI, ISBN 90 6191 7603, 365-367.

PRENN, N.B. (1996). Case studies in open pit design using Lerchs-Grossmann pit optimization. SME Annual Meeting, Phoenix, Arizona, preprint 96-21.
http://www.whittle.com.au

ROBINSON, R.H. et PRENN, N.B. (1972). An open-pit design model. Proc. 10th APCOM, 155-163.

RODITIS, Y.S. (1993). Beyond open pit optimization planning, scheduling and sensitivity analysis. SME Annual Meeting, Reno, Nevada, Frebruary 15-18, preprint 93-227. http://www.whittle.com.au.

ROMAN, R.J. (1974). The role of time value of money in determining an open pit mining sequence and pit limits. Proc. 12th APCOM, Colorado school of mines, Colorado, C72-C85.

RYCHKUN, E.A. et CHEN, T. (1979). Open pit mine feasibility method and application at Placer Development. Computer Methods for the 80's in the Mineral Industry, Weiss A. ed., (New York: AIME), 304-309.

SCHOFIELD, D. et DENBY, B. (1993). Genetic algorithms: A new approach to pit optimisation. Proc. 24th APCOM, 126-133.

SEVIM, H. et LEI, D.D. (1995). Simultaneous determination of open pit mine production planning variables. SME Annual Meeting, Denver, Colorado, March 6-9, Preprint No 95-223.

SEVIM, H. et LEI, D.D. (1996). Production planning with working-slope maximum-metal pit sequences. Trans (Section A: Mining Industry), IMM, 105, A93 - A98.

SEYMOUR, F. (1995). Pit limit parameterization from modified 3D Lerchs-Grossmann algorithm. SME Annual Meeting, Denver, Colorado, March 6-9, Preprint No 95-69.

SEYMOUR, F. (1995). Finding the mining sequence and cutoff grade schedule that maximizes net present value. SME Annual Meeting, Denver, Colorado, March 6-9, Preprint No 95-70.

SIMS, D.E. (1979). Open-pit long-range mine planning using O.R.E. Computer Methods for the 80's in the Mineral Industry. Weiss A.ed., (New York : AIME), 359-370.

STEWART, D.H. (1991). Aspects of pit optimisation, pit design, planning and scheduling. Acads Conference, Kalgoorlie, Australia, 28 november.

http://www.whittle.com.au

STUART, N.J. (1992). Pit optimisation using solid modelling and the Lerchs Grossman algorithm. International Journal of Surface Mining and Reclamation , 6, No 1, 19-29.

TACHEFINE, B.O. (1991). Détermination du plan d'extraction d'une mine à ciel ouvert. Mémoire de maîtrise, Departement de mathématiques appliquées, École Polytechnique de Montréal, Canada, MA91OU, 90p.

TACHEFINE, B.O. (1997). Méthode d'optimisation pour la planification de la production dans les mines à ciel ouvert, Thèse de doctorat, Departement de mathématiques et de génie industriel, École Polytechnique de Montréal, Canada. 150p.

TACHEFINE, B.O. et SOUMIS, F. (1997). Étude comparative des algorithmes de flot maximum pour le problème des contours dans une mine à ciel ouvert. Les cahiers du GERAD (Groupe d'étude en analyse de décisions), G-96-30, 16p.

TACHEFINE, B.O. et SOUMIS, F. (1997). Maximal closure on a graph with resources constraints. Les cahiers du GERAD (Groupe d'étude en analyse de décisions), G-95-44, 18p.

TACHEFINE, B.O., SOUMIS, F. et VANDERSTRATEN, G. (1993). A decomposition flow algorithm for the operations planning problem in open pit mines. Proc. 24th APCOM, 140-147.

TAYLOR, H.K. (1972). General background theory of cutoff grades. Trans. (Section A: Mining Industry), IMM, 81, A160-179.

THOMAS, G.S. (1996). Mine optimization and scheduling – The future. Proc. 1996 Mining Technology Conference, Fremantle, Australia, 85-101.

TOLWINSKI, B. et GOLOSINSKI, T.S. (1995). Long term open pit scheduler. Mine Planning and Equipment Selection, Singhal et al. (eds), Balkema, Rotterdam, ISBN 90 5410 569 0, 265-270.

TOLWINSKI, B. et UNDERWOOD, R. (1992). An algorithm to estimate the optimal evolution of an open pit mine. Proc. 23rd APCOM, 399-409.

TOLWINSKI, B. et UNDERWOOD, R. (1996). A scheduling algorithm for open pit mines. IMA Journal of Mathematics Applied in Business and Industry.

VALLET, R. (1976). Optimisation mathématique de l'exploitation d'une mine à ciel ouvert ou le problème de l'enveloppe. Annales des Mines de Belgique, février, 113-135.

WANG, Q. (1996). Long-term open-pit production scheduling through dynamic phase-bench sequencing. Trans (Section A: Mining Industry), IMM, 105, A99-A104.

WANG, Q. et SEVIM, H. (1992). Enhanced production planning in open pit mining through intelligent dynamic search. Proc. 23rd APCOM, 461-471.

WANG, Q. et SEVIM, H. (1993). An alternative to parameterization in finding a series of maximum-metal pits for production planning. Proc. 24th APCOM, 168-176.

WANG, Q. et SEVIM, H. (1995). Alternative to parameterization in finding a series of maximum-metal pits for production planning. Minnig Engineering, 47, no 2, 178-782.

WHEELER, A. (1997). The shape of things to come at Bjorkdal - Whittle 4D optimisation software at work. Mining Magazine, London, August 1997, 128-130.
http://www.whittle.com.au, E-mail: editorial@mining-journal.com

WHITTLE, J. (1988). Beyond optimization in open pit design. Proc. 1st CAMI, Laval University, Québec, 331-337. http://www.whittle.com.au

WHITTLE, J. (1990). Open pit optimization. Surface Mining, 2nd edition 1990, Society for Mining Metallurgy and Exploration inc., 470-475.
http://www.whittle.com.au

WHITTLE, J. et ROZMAN, L..I. (1991). Open pit design in the 90's. Mining Industry Optimisation Conference, AussIMM, Sydney, June 13-19.
http://www.whittle.com.au

WILKE, F.L., MUELLER, K. et WRIGHT, E.A. (1984). Ultimate pit and production scheduling optimization. Proc. 18th APCOM , London : IMM, 29-38.

WILKE, F.L. et WRIGTH, E.A. (1984) Determining the optimal ultimate pit design for hard rock open pit mines using dynamic programming. Erzmetall, 37, 139-144. (in German)

WILLIAMS, C.E. (1974). Computerized year-by-year open pit mine scheduling. Society of mining Engineers – Transactions, 256, december, 309-317.

WRIGTH, E.A. (1987). The use of dynamic programming for open pit mine design : some practical implications. Mining Science and Technology, 4, 97-104.

WRIGHT, E.A. (1988). Dynamic programming in open pit mining sequence planning - A case study. Proc., 21th APCOM, 415-422.

WRIGHT, E.A. (1999). Moving cone II - A simple algorithm for optimum pit limits design. Proc., 28th APCOM, 367-374.

WRIGHT, E.A., MBIRIKIRA, D.S.B. et DUBE, T.Y. (1993). Recent findings in open pit optimization. International Journal of Surface Mining and Reclamation , 7, No 4, 155-159.

YAMATOMI, J., MOGI, G., AKAIKE, A. et YAMAGUCHI, U. (1995). Selective extraction dynamic cone algorithm for tree-dimensional open-pit designs. Proc. 25th APCOM, 267-274.

YEGULALP, T.M. et ARIAS, J.A. (1992). A fast algorithm to solve the ultimate pit limit problem. Proc. 23rd APCOM, 391-397.

ZHANG, Y. et YANG, R. (1990). Optimization of open pit limit with dynamic-economy analysis. International Journal of Surface Mining and Reclamation , 4, No 1, 21-24.

ZHAO, Y. et KIM, Y.C. (1990). A new graph theory algorithm for optimal pit design. SME Annual Meeting, Salt Lake City, Utah, U.S.A., Preprint No. 90-9.

ZHAO, Y. et KIM, Y.C. (1992). A new optimum pit limit design algorithm. Proc. 23rd APCOM, 423-434.

ZHAO, Y. and KIM, Y.C. (1993). Optimum mine production sequencing using lagrangian parameterization approach. Proc. 24th APCOM, 176-183.

Adresses internet:

http://www.whittle.com.au (informations et articles sur Whittle)

http://archi.snu.ac.kr/bwkim/free_or_codes.html (code du flot maximum)

http://www.surpac.com (informations sur le logiciel Surpac)

http://riot.ieor.berkeley.edu/~dorit/pub/mine.html (article de Hochbaum et Chen)

ANNEXE I

LERCHS ET GROSSMANN 2D

Voici les étapes de l'algorithme de Lerchs et Grossmann 2D:

1. Tout d'abord, on doit établir la matrice à deux dimensions qui contient la valeur nette de chacun des blocs. Soit m_{ij} la valeur nette du bloc situé à l'intersection du niveau i et de la colonne j. De cette matrice, on calcule la valeur M_{ij} de chaque bloc.

$$M_{ij} = \sum_{k=1}^{i} m_{kj} \qquad \text{où} \quad i = 1, \ldots, I \quad \text{et} \quad j = 1, \ldots, J$$

   En fait M_{ij} représente le profit réalisé lorsqu'on extrait une colonne dont la base correspond à l'élément (i, j).

2. a) Ensuite on ajoute un niveau artificiel, noté i = 0, où la valeur $m_{0j} = 0$ et ce pour j = 1, …., J.

b) On initialise les variables j = 0 et $P_{0j} = 0$.

3. Si j = J on se rend immédiatement à l'étape 6.

4. On pose j = j + 1.

5. a) $P_{ij} = M_{ij} + \max (P_{i+r,\, j-1})$ où $k \in \{-1, 0, 1\}$

   On indique comment la valeur maximale a été obtenue pour chaque bloc (i, j) par une flèche du bloc (i, j) au bloc $(i + r^*, j - 1)$ où r^* est la valeur pour laquelle P_{ij} est maximale. P_{ij} représente la contribution maximale des colonnes 1 à j de toute extraction qui inclurait le bloc (i, j) sur ses contours ultimes.

b) On retourne à l'étape 3.

6. P_{0J} représente le profit net total, optimal pour cette matrice et le profit miné.

Si $P_{0J} > 0$, alors on effectue un tracé en suivant les flèches du bloc (0, J) de la colonne J vers la colonne 0 jusqu'au bloc (0, 0). Le tracé indique le contour optimal du volume miné pour cette session.

ANNEXE II
LERCHS ET GROSSMANN 3D

Tout d'abord, afin de bien comprendre la méthode, on doit donner quelques définitions.

Définition : Une *fermeture* sur un graphe G = (N, A) est l'ensemble des nœuds $Y \in N$ tels que tous les nœuds $i \in Y$ ayant tous ses successeurs appartenant à Y.

Définition : La *racine* s d'un arbre orienté, noté T, est un nœud sans prédécesseur.

Définition : Un graphe obtenu en éliminant un arc (i, j) d'un arbre contient deux éléments. L'élément qui ne contient pas la racine est appelé une *branche*, noté T_j. On dit que l'arc (i, j) supporte la branche T_i.

Définition : Si le nœud terminal de l'arc (i, j) appartient à la branche T_j, alors on dit que l'arc (i, j) est un *arc-p* ou encore que T_j est une *branche-p*.

Définition : Si le nœud terminal de l'arc (i, j) n'appartient pas à la branche T_j, alors on dit que l'arc (i, j) est un *arc-m* ou encore que T_j est une *branche-m*. Si on associe un poids ou une valeur m_i à tous les nœuds $i \in N$, alors M_i est la somme des valeurs des nœuds de la branche:

$$M_i = \sum_{i \in T_i} m_i$$

Définition : Un *arc* est *fort* s'il supporte une branche-p dont les $M_i > 0$ ou s'il supporte une branche-m dont le $M_i \leq 0$.

Définition : Un *arc* est *faible* s'il n'est pas fort.

Définition : Un *nœud* k est *fort* s'il existe au moins un arc fort sur la chaîne de T entre la racine et le nœud k, sinon il est dit *faible*.

L'algorithme de Lerchs et Grossmann est séparé en deux parties. La première partie sert à construire le graphe et la deuxième partie sert à calculer la fermeture maximale sur le graphe.

Lors de la première partie, on construit le graphe initial en traçant des arcs orientés entre chacun des nœuds et les 9 nœuds qui se trouvent au-dessus de lui. Par exemple supposons que nous avons le gisement en deux dimensions présenté à la figure A2.1:

-2	-2	-2	-2	-2	3	-2	-2
-2	-2	8	-2	-2	-2	-2	-2
-2	-2	6	-2	5	-2	-2	-2

Figure A2.1 : Exemple de gisement

On peut associer à ce gisement le graphe des préséances présenté à la figure A2.2:

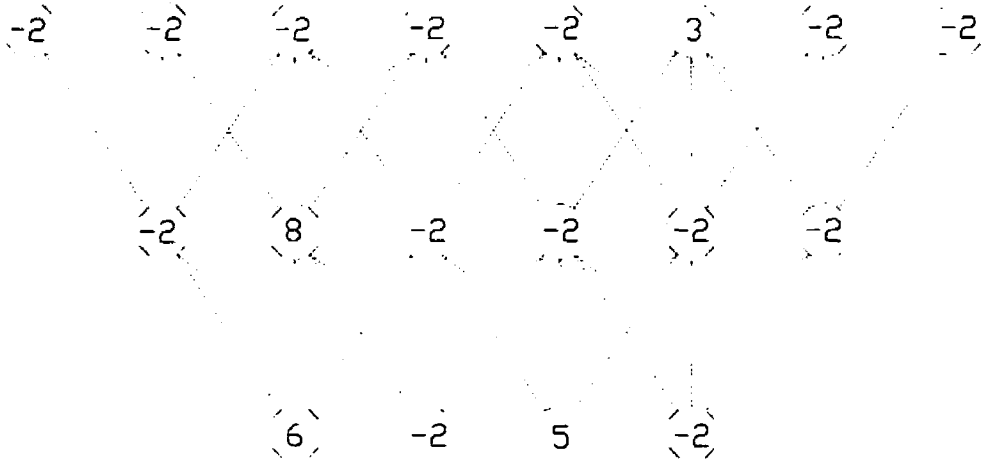


Figure A2.2 : Graphe représentant les contraintes de préséance entre les blocs

Pour terminer, on doit relier tous les nœuds à un nœud artificiel nommé s. On forme ainsi un arbre où le nœud artificiel s est la racine de l'arbre. La figure A2.3 présente cet arbre initial.

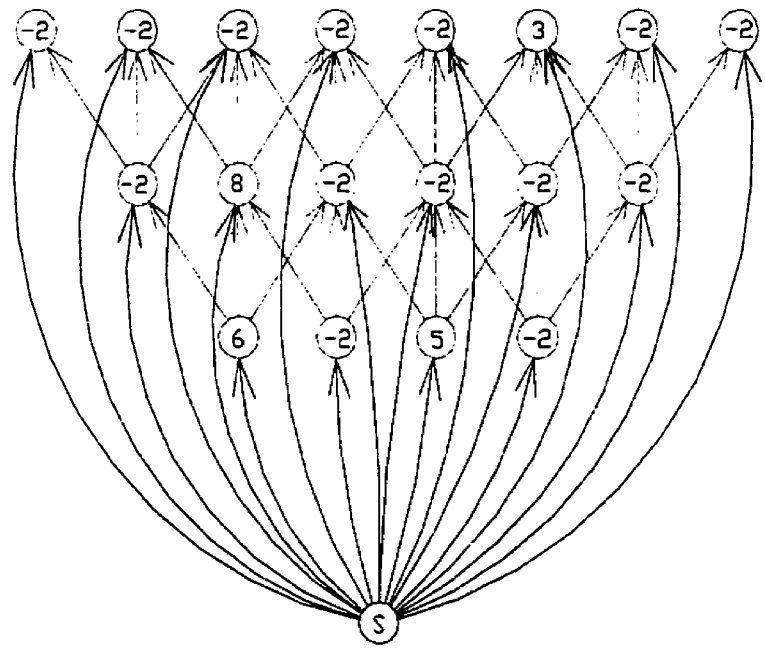


Figure A2.3 : Arbre initial du graphe de Lerchs et Grossmann

La deuxième partie de l'algorithme consiste à appliquer l'algorithme au graphe qui a été déterminé dans la première partie. Soit les définitions suivantes:

Définition : Un *arbre* est dit *normalisé* si la racine s est commune à tous les à tous les arcs forts. Tout arbre T d'un graphe G(N, A) peut être normalisé en remplaçant l'arc-p fort (s, k) par un arc bidon (s, l) ou un arc-m faible (q, r) par un arc bidon (s, q) et en répétant le processus jusqu'à ce que les arcs forts aient s comme extrémité.

Preuve 1 : Si un nœud k appartient à Z, la fermeture maximale d'un arbre T normalisé, alors tous les nœuds de l'ensemble T_k, la branche ayant comme racine k, appartiennent aussi à Z.

Preuve 2 : La fermeture maximale d'un arbre normalisé T est l'ensemble Z de ses sommets forts.

Théorème : Dans un graphe direct G(N, A), un arbre normalisé T peut être construit tel que l'ensemble Y des sommets forts de T est une fermeture de G, alors Y est une fermeture maximale de G.

Pour l'algorithme qui suit, on construit un arbre normalisé T^0 dans G et on procède par la suite par itérations. À l'itération t+1 on transforme l'arbre normalisé T^t en un nouvel arbre normalisé T^{t+1}. Chaque arbre $T^t(N, A^k)$ est caractérisé par un ensemble d'arcs A^t et un ensemble de nœuds forts Y^t. L'algorithme se termine lorsque Y^t est vide.

Algorithme de Lerchs et Grossmann 3-D:

1. Tous les nœuds de G(N, A) sont reliés par un arc à une racine artificielle pour former un arbre noté T^0. Cet arbre est normalisé.

2. Les arcs sont désignés forts ou faibles.

3. Les nœuds forts formant une fermeture sont enlevés de T^t.

4. Par un processus de normalisation, les nœuds situés sur des arcs forts sont séparés du nœud racine s et deviennent la racine d'une branche constituée de tous les nœuds de la fermeture associée à ce nœud. Si la branche devient forte on l'enlève, sinon elle demeure. Le processus s'effectue comme suit:

5. Tant qu'il existe un nœud $k \in Y^t$ et un nœud $l \in N \backslash Y^t$, alors:

   - déterminer *m* la racine de la branche forte contenant k;
   - construire T^s en remplaçant (0, m) de T^t par (k, l).

Le processus de normalisation continue jusqu'à ce que tous les ensembles reliés à la racine artificielle soient faibles. L'ensemble de nœuds restants définit la forme de la fosse et la valeur correspond à la somme des nœuds enlevés.

ANNEXE III

«LISTING» COMPLET DU PROGRAMME

Cette annexe comprend le code des programmes créés pour les fins du mémoire et qui utilisés pour les trois séries de tests. La première partie présente le programme de base puis dans les trois autres parties on ne présente seulement que les procédures qui changent par rapport au programme de base.

A3.1 «Param.cpp» ou «Param_ninv.cpp»

En première partie, on présente le «listing» complet du programme qui calcule les contours ultimes d'une fosse à ciel ouvert à l'aide d'un algorithme de flot maximum. Il s'agit du programme original qui calcul des coûts pour chacun des blocs à partir des teneurs et pour un prix du métal donné par l'usager. Il crée ensuite le graphe de Picard, pour le flot maximum, en reliant le nœud source à tous les nœuds qui ont une valeur positive et en reliant tous les nœuds qui ont une valeur négative au nœud puits. Il s'agit du programme nommé «Param.cpp» ou encore «Param_ninv.cpp» dans le mémoire.

«listing»:

```
/* PARTIE INCLUSION DES FICHIERS GLOBALS */

#include <fstream.h>          //Pour l'utilisation de ifstream et ofstream
#include <iostream.h>         //Pour l'utilisation de cin et cout
#include <iomanip.h>          //Pour l'utilisation de setw() et setprecision()
#include <stdlib.h>           //Pour l'utilsation de system()

#define max_periodes 50       //bon pour un maximum de une fosse
#define non 0
#define oui 1
#define infini 100000000

struct blocs
        {
        int coor_x;
        int coor_y;
        int coor_z;
        float litho;
```

```
        float teneur;
        float wrec;
        float densite;
        int cout[max_periodes];
        int predecesseurs[9];
        int eliminer;
        int periode_extraction;
        };


typedef char type_nom[50];

type_nom nom_fichier;
fstream fichier;

ifstream fichier_a_lire;

/*-------------------------------------------------------------------------  */
/* SOUS-PROGRAMME : INITIALISATION                                          */
/* DESCRIPTION : Ce programme est une sous-routine qui est appelée chaque*/
/* fois qu'une variable de type blocs doit être initialisée                 */
/*-------------------------------------------------------------------------  */
void initialisation(blocs & bloc)
                //Le paramètre est transmis par adresse dans cette fonction.
        {
                /* PARTIE DÉCLARATION */
        int i;

                /*PARTIE INSTRUCTIONS */

        bloc.litho = 0;
        bloc.teneur = 0;
        bloc.wrec = 0;
        bloc.densite = 0;
        bloc.coor_x = 0;
        bloc.coor_y = 0;
        bloc.coor_z = 0;
        for (i = 0; i < max_periodes; i++)
                bloc.cout[i] = 0;
        for (i = 0; i < 9; i++)
                bloc.predecesseurs[i] = 0;
        bloc.eliminer = non;
        bloc.periode_extraction = 0;
        }

/*-------------------------------------------------------------------------  */
/* SOUS-PROGRAMME : LECTURE_LIGNE*/
/* DESCRIPTION : Ce programme est une sous-routine qui permet de lire une ligne*/
/* du fichier initial de Surpac.  Le fichier d'input doit se présenter sous la */
/* forme suivante: String, y, x, z, litho, teneur, wrec, densite, rock_code, mcaf, */
/* pcaf.  Le fichier ne doit comprendre aucune en-tête, ni de message pour */
/* indiquer la fin du fichier.*/
```

```
/*-------------------------------------------------------------------------- */
void lecture_ligne(float& coor_y, float& coor_x, float& coor_z, float& lito,
                   float& ten, float& wr, float& dens)

                   //Les paramètres sont transmis par adresse dans cette fonction.
        {

                   /* PARTIE DÉCLARATION DES VARIABLES */

        float r_c, mf, pf;
        char e;
        int string;

                   /* PARTIE INITIALISATION DES VARIABLES */

        ten = 0;
        wr = 0;
        dens = 0;
        string = 0;
        coor_y = 0;
        coor_x = 0;
        coor_z = 0;
        lito = 0;
        r_c = 0;
        mf = 0;
        pf = 0;

                   /* PARTIE INSTRUCTION */

        fichier_a_lire >> string;
        fichier_a_lire >> e;
        fichier_a_lire >> coor_y;
        fichier_a_lire >> e;
        fichier_a_lire >> coor_x;
        fichier_a_lire >> e;
        fichier_a_lire >> coor_z;
        fichier_a_lire >> e;
        fichier_a_lire >> lito;
        fichier_a_lire >> e;
        fichier_a_lire >> ten;
        fichier_a_lire >> e;
        fichier_a_lire >> wr;
        fichier_a_lire >> e;
        fichier_a_lire >> dens;
        fichier_a_lire >> e;
        fichier_a_lire >> r_c;
        fichier_a_lire >> e;
        fichier_a_lire >> mf;
        fichier_a_lire >> e;
        fichier_a_lire >> pf;
        }
```

```
/*-------------------------------------------------------------------------  */
/* SOUS-PROGRAMME : OUI_NON*/
/* DESCRIPTION : Ce programme est une sous-routine qui demande une réponse*/
/* affirmative ou négative et de validadr l'entrée.*/
/*-------------------------------------------------------------------------  */
void oui_non(int& reponse)
        //Le paramètre est transmis par adresse dans cette fonction.
    {
        /*PARTIE INSTRUCTIONS */
    cout <<"1 = oui, 2 = non : ";
    cin >> reponse;
    cout << endl;
    while ((reponse != 1) && (reponse != 2))
        {
        cout << "Recommencez s.v.p. , 1 = oui, 2 = non  : ";
        cin >> reponse;
        cout << endl;
        }
    }

/*-------------------------------------------------------------------------*/
/*SOUS-PROGRAMME: nouvelle_borne   */
/*DESCRIPTION:Il s'agit d'un sous_programme qui recalcul les nouvelles bornes */
/* lorsqu'une réduction de fichier est demandé par l'utilisateur.  J'ai inséré */
/* ce sous-programme pour ne pas avoir à relire le fichier afin de déterminer */
/* les nouvelles bornes qui sont nécessaires dans tous les calculs au cas ou  */
/* l'utilisateur n'entrerais les bornes juste....  */
/*-------------------------------------------------------------------------*/
void nouvelle_borne(int& min, int& max, int inferieur, int superieur,
        int nombre, int pas)
    {
        /*PARTIE DÉCLARATION DES VARIABLES */
    int i;
    int bidon_min, bidon_max, min_temp, max_temp;

        /* PARTIE INITIALISATION DES VARIABLES */

    min_temp = 0;
    max_temp =0;

        /* PARTIE INSTRUCTIONS */
    for ( i = 0; i < nombre; i++)
        {
        bidon_min = min + i*pas;
        bidon_max = max - i*pas;
        if (bidon_min < inferieur)
            min_temp = bidon_min;
        if (bidon_max > superieur)
            max_temp = bidon_max;
        }
    min = min_temp + pas;
    max = max_temp - pas;
```

```
        }

/*-------------------------------------------------------------------------*/
/*SOUS-PROGRAMME: donnee_reduction                                         */
/*DESCRIPTION:Il s'agit d'un sous_programme qui demande les bornes supérieures */
/* et inférieures lorsqu'on désire résuire le gisement et qui recalcule */
/* automatiquement les nouvelles bornes réelles du gisement.*/
/*-------------------------------------------------------------------------*/
void donnee_reduction(int& lig_min, int& lig_max, int& col_min, int& col_max,
        int& niv_min, int& niv_max, int& nbr_niveau, int& nbr_ligne,
        int& nbr_colonne, int& nbr_blocs, int prof, int haut, int larg)

        {

                /* PARTIE DÉCLARATION DES VARIABLES */

        int inferieur_x, superieur_x, inferieur_y, superieur_y, inferieur_z, superieur_z;

                /* PARTIE INITIALISATION DES VARIABLES */

        inferieur_x = 0;
        superieur_x = 0;
        inferieur_y = 0;
        superieur_y = 0;
        inferieur_z = 0;
        superieur_z = 0;


                /* PARTIE INSTRUCTIONS */

        cout << "Vous devez me donner les nouvelles bornes du gisement: "<<endl;
        cout << "Borne inferieure en x : ";
        cin >> inferieur_x;
        cout << "Borne superieure en x : ";
        cin >> superieur_x;
        cout << "Borne inferieure en y : ";
        cin >> inferieur_y;
        cout << "Borne superieure en y : ";
        cin >> superieur_y;
        cout << "Borne inferieure en z : ";
        cin >> inferieur_z;
        cout << "Borne superieure en z : ";
        cin >> superieur_z;

        nouvelle_borne(niv_min, niv_max, inferieur_z, superieur_z, nbr_niveau, haut);
        nouvelle_borne(lig_min, lig_max, inferieur_x, superieur_x, nbr_ligne, prof);
        nouvelle_borne(col_min, col_max, inferieur_y, superieur_y, nbr_colonne, larg);

        nbr_niveau = ((niv_max - niv_min) / haut) + 1;
        nbr_ligne = ((lig_max - lig_min) / prof) + 1;
        nbr_colonne = ((col_max - col_min) / larg) + 1;
        nbr_blocs = nbr_niveau * nbr_ligne * nbr_colonne;
```

```
        cout << "Votre fichier aura " << nbr_blocs << " blocs." << endl;
        }

/*-------------------------------------------------------------------------  */
/* SOUS-PROGRAMME : LECTURE*/
/* DESCRIPTION : Ce programme est une sous-routine du programme principal*/
/* Il permet de lire les données à partir du fichier de initial de Surpac.*/
/* Le fichier d'input doit se présenter sous la forme suivante:*/
/*        String, y, x, z, litho, teneur, wrec, densite.*/
/* Le fichier ne doit comprendre aucune en-tête, ni de message pour */
/* indiquer la fin du fichier.*/
/* Le programme demande le système de référence du gisement, donne l'origine*/
/* et l'étendue du gisement (Ces données sont nécessaire pour SURPAC).*/
/* Ce programme permet en plus de déterminer combien on a de lignes, de colonnes et  */
/* de niveau pour pouvoir par la suite numeroter les blocs correctement pour */
/* pour l'optimisation.*/
/*-------------------------------------------------------------------------  */
void lecture(type_nom fichier_lecture, int& nbr_bloc, int& nbr_n, int& nbr_l,int& nbr_c,
        int& l_min, int& l_max, int& c_min, int& c_max, int& n_min, int& n_max,
        int& prof, int& larg, int& haut)

        //Tous les paramètres sont transmis par adresse dans cette fonction.

        {
                /* PARTIE DÉCLARATION */

        float teneur, wrec, densite, coordonne_x, coordonne_y, coordonne_z, litho;
        int compteur;
        int bidon_ligne, bidon_colonne, bidon_niveau;
                //lignes et prof = x, colonnes et larg = y, niveaux et haut =z

                /* INITIALISATION DES VARIABLES */

        bidon_ligne = infini;
        bidon_niveau = infini;
        bidon_colonne = infini;
        coordonne_x = 0;
        coordonne_y = 0;
        coordonne_z = 0;
        compteur = 0;

                /*PARTIE INSTRUCTIONS */

        fichier_a_lire.open(fichier_lecture);          //ouverture du fichier en mode lecture
        if ( fichier_a_lire.fail())
                cout << "(1) Impossible d'ouvrir "<< fichier_lecture <<endl;
        else
                {
                while (!fichier_a_lire.eof())
                        {
                        compteur++;
```

```
                    lecture_ligne(coordonne_y, coordonne_x, coordonne_z, litho, teneur, wrec,
                    densite);

                    l_min = bidon_ligne ;
                    c_min = bidon_colonne ;
                    n_min = bidon_niveau ;

                    if (coordonne_x < bidon_ligne )
                            bidon_ligne  = coordonne_x;
                    if (coordonne_x > l_max)
                            l_max = coordonne_x;
                    if (coordonne_y < bidon_colonne)
                            bidon_colonne = coordonne_y;
                    if (coordonne_y > c_max)
                            c_max = coordonne_y;
                    if (coordonne_z < bidon_niveau)
                            bidon_niveau = coordonne_z;
                    if (coordonne_z > n_max)
                            n_max = coordonne_z;
                    }
            compteur--;
            cout << "Les maximums sont : "<<l_max<<" en x, "<<c_max<<" en y et "<<n_max<<"
en z"<<endl;
            cout << "Les minimums sont : "<<l_min<<" en x, "<<c_min<<" en y et "<<n_min<<"
en z"<<endl;
            cout << "L'etendue centroide à centroide du gisement est : "<<(l_max-l_min)<<" en x, ";
            cout << (c_max - c_min)<<" en y et "<<(n_max - n_min)<<" en z"<<endl;
            cout << "Entrez les dimensions des blocs : "<<endl;
            cout << " y = ";
            cin >> larg;
            cout << " x = ";
            cin >> prof;
            cout << " z = ";
            cin >> haut;
            cout << endl;
            while(((c_max - c_min)%larg) + ((l_max-l_min)%prof) + ((n_max - n_min)%haut) != 0)
                    {
                    cout<<"Vous vous etes trompe, recommencez."<<endl;
                    cout << "Entrez les dimensions des blocs : "<<endl;
                    cout << " y = ";
                    cin >> larg;
                    cout << " x = ";
                    cin >> prof;
                    cout << " z = ";
                    cin >> haut;
                    cout << endl;
                    }

            nbr_n = ((n_max - n_min) / haut) + 1;
            nbr_l = ((l_max - l_min) / prof) + 1;
            nbr_c = ((c_max - c_min) / larg) + 1;
```

```
                nbr_bloc = nbr_n * nbr_l * nbr_c;
                cout << "Votre gisement comprend "<< compteur <<" blocs. "<<endl;
                if (nbr_bloc != compteur)
                        {
                        cout<<"Votre gisement n'est pas complet"<<endl;
                        cout<<"Il manque "<< (nbr_bloc - compteur) << " blocs."<<endl;
                        }
                }
        fichier_a_lire.close();
        }

/*-------------------------------------------------------------------------*/
/*SOUS-PROGRAMME: cas_1_1                                                  */
/*DESCRIPTION:C'est ce programme qui est appelé si on désire résuire le gisement */
/* et l'optimiser tout de suite après.  Le fichier qui sera lu et réduit ici est  */
/* le même que celui qui a été lu dans la partie lecture du programme.            */
/*-------------------------------------------------------------------------*/

void cas_1_1(type_nom fichier_lecture, int& nbr_bloc, int& nbr_n, int& nbr_l, int& nbr_c,
                int& l_min, int& l_max, int& c_min, int& c_max, int& n_min, int& n_max,
                int prof, int larg, int haut, int reference)

        {

                /* PARTIE DÉCLARATION DES VARIABLES */

        float teneur, wrec, densite, coordonne_y, coordonne_x, coordonne_z, litho;
        int x, y, z;

        int no_bloc;

        blocs bloc;

                /* PARTIE INITIALISATION DES VARIABLES */

        initialisation(bloc);

        coordonne_y = 0;
        coordonne_x = 0;
        coordonne_z = 0;
                /* PARTIE INSTRUCTION */

        donnee_reduction(l_min, l_max, c_min, c_max, n_min, n_max, nbr_n, nbr_l,
                nbr_c, nbr_bloc, prof, haut, larg);

        fichier_a_lire.open(fichier_lecture);          //ouverture du fichier en mode lecture
        if ( fichier_a_lire.fail())
                cout << "(1) Impossible d'ouvrir "<< fichier_lecture <<endl;
        else
                {
                while (!fichier_a_lire.eof())
                        {
```

```
                    if((coordonne_x >= l_min) && (coordonne_x <= l_max) &&
                    (coordonne_y >= c_min) && (coordonne_y <= c_max) &&
                    (coordonne_z >= n_min) && (coordonne_z <= n_max))
                            {
                            if (reference == 1)          //Si le systeme de reference est vers le haut
                                    {
                                    x = ((coordonne_x - l_min) / prof) + 1;
                                    y = ((coordonne_y - c_min) / larg) + 1;
                                    z = ((n_max - coordonne_z)/ haut) + 1;
                                    no_bloc = y + (x - 1)*(nbr_c) + (z - 1)*(nbr_c)*(nbr_l);
                                    }
                            if (reference == 2)          //Si le systeme de reference est vers le bas
                                    {
                                    x = ((coordonne_x - l_min) / prof) + 1;
                                    y = ((coordonne_y - c_min) / larg) + 1;
                                    z = ((coordonne_z - n_min) / haut) + 1;
                                    no_bloc = x + (y - 1)*(nbr_l) + (z - 1)*(nbr_l)*(nbr_c);
                                    }
                            fichier.seekp((no_bloc - 1)*sizeof(blocs), ios::beg);
                            fichier.write((char*) & bloc, sizeof(bloc));
                            }
                    initialisation(bloc);
                    lecture_ligne(coordonne_y, coordonne_x, coordonne_z, litho, teneur, wrec,
                            densite);
                    bloc.coor_x = coordonne_x;
                    bloc.coor_y = coordonne_y;
                    bloc.coor_z = coordonne_z;
                    bloc.litho = litho;
                    bloc.teneur = teneur;
                    bloc.wrec = wrec;
                    bloc.densite = densite;
                    }
            }
            fichier_a_lire.close();
    }

/*-------------------------------------------------------------------*/
/*SOUS-PROGRAMME: cas_1_2                                            */
/*DESCRIPTION:C'est ce programme qui est appelé si on désire résuire le gisement */
/* mais ne pas procédé à l'optimisation tout de suite après.  Le fichier qui sera */
/* lu et réduit ici est       le même que celui qui a été lu dans la partie lecture */
/* du programme.                                                     */
/*-------------------------------------------------------------------*/

void cas_1_2(type_nom fichier_lecture, int prof, int haut, int larg, int l_min,
            int l_max, int c_min, int c_max, int n_min, int n_max, int nbr_l,
            int nbr_c, int nbr_n, int nbr_bloc)
    {
            /*PARTIE DÉCLARATION DES VARIABLES*/

    float teneur, wrec, densite, coordonne_y, coordonne_x, coordonne_z, litho;
    int string, rock_code, mcaf, pcaf;
```

```
char fichier_ecriture[50];

ofstream fichier_a_ecrire;

        /* PARTIE INSTRUCTIONS */

donnee_reduction(l_min, l_max, c_min, c_max, n_min, n_max, nbr_n, nbr_l,
                        nbr_c, nbr_bloc, prof, haut, larg);

cout << "Donnez-moi le nom de votre nouveau fichier réduit : "<<endl;
cin >> fichier_ecriture;
fichier_a_ecrire.open(fichier_ecriture);       //ouverture du fichier en mode ecriture
if ( fichier_a_ecrire.fail())
        cout<<"(3) Impossible d'ouvrir " << fichier_ecriture << endl;
else
        {
        fichier_a_lire.open(fichier_lecture);          //ouverture du fichier en mode lecture
        if ( fichier_a_lire.fail())
                cout << "(1) Impossible d'ouvrir "<< fichier_lecture <<endl;
        else
                {
                while (!fichier_a_lire.eof())
                        {
                        lecture_ligne(coordonne_y, coordonne_x, coordonne_z, litho,
                                teneur, wrec, densite);

                        if((coordonne_x >= l_min) && (coordonne_x <= l_max) &&
                        (coordonne_y >= c_min) && (coordonne_y <= c_max) &&
                        (coordonne_z >= n_min) && (coordonne_z <= n_max))
                                {
                                string = 1;
                                if ((litho == 99)||(litho == 19))
                                        {
                                        rock_code = 99;
                                        mcaf = 0;
                                        pcaf = 0;
                                        }
                                if (litho == 103)
                                        {
                                        rock_code = 103;
                                        mcaf = 1;
                                        pcaf = 1;
                                        }
                                if ((litho == 100)||(litho == 101)||(litho == 102)||(litho == 104)
                                ||(litho == 106)||(litho == 108)||(litho == 202)||(litho == 203))
                                        {
                                        rock_code = 100;
                                        mcaf = 1;
                                        pcaf = 0;
                                        }
                                fichier_a_ecrire << string <<" , "<< coordonne_y << " , " <<
coordonne_x;
```

```
                                        fichier_a_ecrire  <<" , "<< coordonne_z <<" , "<< litho << " ,
" ;
                                        fichier_a_ecrire << teneur <<" , "<< wrec <<" , "<< densite;
                                        fichier_a_ecrire  <<"  , "<<rock_code<<"  , "<<mcaf<<"  ,
"<<pcaf<< endl;
                                        }
                                }
                        }
                fichier_a_lire.close();
                }
        fichier_a_ecrire.close();
        }

/*-------------------------------------------------------------------------*/
/*SOUS-PROGRAMME: cas_2_1                                                  */
/*DESCRIPTION:C'est ce programme qui est appelé si on ne désire pas résuire le */
/* gisement mais seulement l'optimiser.  Le fichier qui sera lu est        */
/* le même que celui qui a été lu dans la partie lecture du programme.  */
/*-------------------------------------------------------------------------*/

void cas_2_1(type_nom fichier_lecture, int nbr_l, int nbr_c, int l_min, int c_min,
        int n_min, int n_max, int prof, int larg, int haut, int reference)
        {

                /* PARTIE DÉCLARATION DES VARIABLES */

        float teneur, wrec, densite, coordonne_y, coordonne_x, coordonne_z, litho;
        int x, y, z;

        int no_bloc;

        blocs bloc;

                /* PARTIE INITIALISATION DES VARIABLES */

        no_bloc = 1;
        initialisation(bloc);

                /* PARTIE INSTRUCTION */

        fichier_a_lire.open(fichier_lecture);          //ouverture du fichier en mode lecture
        if ( fichier_a_lire.fail())
                cout << "(1) Impossible d'ouvrir "<< fichier_lecture <<endl;
        else
                {
                while (!fichier_a_lire.eof())
                        {
                        fichier.seekp((no_bloc - 1)*sizeof(blocs), ios::beg);
                        fichier.write((char*) & bloc, sizeof(bloc));
                        initialisation(bloc);

                        lecture_ligne(coordonne_y, coordonne_x, coordonne_z, litho, teneur, wrec,
```

```
                densite);

            if (reference == 1)          //Si le systeme de reference est vers le haut
                {
                x = ((coordonne_x - l_min) / prof) + 1;
                y = ((coordonne_y - c_min) / larg) + 1;
                z = ((n_max - coordonne_z)/ haut) + 1;
                no_bloc = y + (x - 1)*(nbr_c) + (z - 1)*(nbr_c)*(nbr_l);
                }
            if (reference == 2)          //Si le systeme de reference est vers le bas
                {
                x = ((coordonne_x - l_min) / prof) + 1;
                y = ((coordonne_y - c_min) / larg) + 1;
                z = ((coordonne_z - n_min) / haut) + 1;
                no_bloc = x + (y - 1)*(nbr_l) + (z - 1)*(nbr_l)*(nbr_c);
                }
            bloc.coor_x = coordonne_x;
            bloc.coor_y = coordonne_y;
            bloc.coor_z = coordonne_z;
            bloc.litho = litho;
            bloc.teneur = teneur;
            bloc.wrec = wrec;
            bloc.densite = densite;
            }
        }
    fichier_a_lire.close();
    }


/*-------------------------------------------------------------------------*/
/* SOUS-PROGRAMME : CALCUL_COUT                                            */
/* DESCRIPTION :    Ce sous-programme est appelé par le programme principal */
/* Il calcul les couts pour chacun des blocs et pour toutes les périodes. */
/* Il demande tout d'abord le prix du métal, le nombre de périodes voulue (- de 50) */
/* puis il calcul des couts pour chacune des périodes.                     */
/*-------------------------------------------------------------------------*/
void calcul_cout(int& nbr_per, int nbr_bloc, int larg, int haut, int prof)

    {
        /* PARTIE DÉCLARATION */

    blocs bloc;

    int i, j;
    float prix, prix_inf, prix_sup, minage, traitement, transport;
    float pas;

        /* PARTIE INSTRUCTIONS */

    cout << "Combien de periodes voulez-vous? (maximum de 50) : ";
    cin >> nbr_per;
    cout << endl;
```

```
while ((nbr_per < 1) && (nbr_per > 50))
        {
        cout << "Recommencez s.v.p. , (maximum 50 periodes) : ";
        cin >> nbr_per;
        cout << endl;
        }

if (nbr_per == 1)
        {
        cout << "Quel est le prix de vente ($/tonne de concentre) : ";
        cin >> prix;
        cout << endl;
        }
if (nbr_per > 1)
        {
        cout <<"Quel est la borne inferieur du prix de vente ($/tonne de concentre) : ";
        cin >> prix_inf;
        cout <<"Quel est la borne superieur du prix de vente ($/tonne de concentre) : ";
        cin >> prix_sup;
        while (prix_sup < prix_inf)
                {
                cout <<"Recommencez s.v.p." << endl;
                cout <<"Quel est la borne inferieur du prix de vente ($/tonne de concentre) : ";
                cin >> prix_inf;
                cout <<"Quel est la borne superieur du prix de vente ($/tonne de concentre) : ";
                cin >> prix_sup;
                cout << endl;
                }
        pas = (prix_sup - prix_inf)/(nbr_per - 1);
        }

cout << "Quel est le cout de minage du stérile et du minerai ($/tonne de roche) : ";
cin >> minage;
cout << "Quel est le cout de traitement ($/tonne de minerai) : ";
cin >> traitement;
cout << "Quel est le cout de transport ($/tonne de concentre) : ";
cin >> transport;
cout << endl;

for ( i = 1; i <= nbr_bloc; i++)
        {
        initialisation(bloc);
        fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
        fichier.read((char*) & bloc, sizeof(bloc));
        if (bloc.eliminer == non)
                {
                for (j = 0; j < nbr_per; j++)
                        {
if (bloc.teneur > 0)
                                {
                                if (nbr_per > 1)
```

```
                                        bloc.cout[j] = (bloc.densite*(larg*haut*prof)*((0.95
*((prix_inf + j*pas) - transport)* bloc.wrec/66.6)-(minage + traitement)));
                                    else
                                        bloc.cout[j] = (bloc.densite * (larg*haut*prof) *
((0.95 *(prix - transport)* bloc.wrec/66.6)-(minage + traitement)));
                                    }
                                else
                                    bloc.cout[j] = - (bloc.densite * (larg*haut*prof) * minage);
                                }
    }
            fichier.seekp((i - 1)*sizeof(blocs), ios::beg);
            fichier.write((char*) & bloc, sizeof(bloc));
  }
      }


/*-------------------------------------------------------------------*/
/* SOUS-PROGRAMME : PRESEANCE_PARTIE1
      */
/* DESCRIPTION :      Ce sous-programme est appelé par le programme principal.*/
/* Il sert à transformer les données lues dans le fichier texte de façon à */
/* pouvoir créer le graph avec lequel le calcul du flow max se fera.        */
/* Cette première partie sert à éliminer les blocs qui nous sont fournis et  */
/* qui ne pourront jamais faire partis du graph. Soit les blocs qui ne       */
/* font pas partis du cône d'extraction et qui ne respectent pas les pentes  */
/* des talus. Donc tout d'abord le programme identifie les blocs qui se */
/* trouvent au premier niveau. Ces blocs doivent subir un traitement         */
/* spécial parce qu'ils n'ont pas de prédecesseur mais qu'ils peuvent tous   */
/* être extraits. Par la suite il élimine tous les blocs qui ne respectent  */
/* pas les pentes des talus.                                                 */
/*-------------------------------------------------------------------  */

void preseance_partie1(int nbr_n, int nbr_l, int nbr_c)
      {
            /* PARTIE DÉCLARATION */

      blocs bloc;
      int i, n, c, l;            //les compteurs

            /* PARTIE INSTRUCTIONS */

      for ( n = 2; n <= nbr_n; n++)
            for (c = 1; c <= nbr_c; c++)
                  for (l = 1; l <= nbr_l; l++)
                        {
                        i = (n-1)*nbr_l*nbr_c + (c - 1)*nbr_l + l;
                        initialisation(bloc);
                        fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
                        fichier.read((char*) & bloc, sizeof(bloc));
                        if ((c >= n)&&(c <= nbr_c - n + 1)&&
                              (l >= n)&&(l <= nbr_l - n + 1))
                              bloc.eliminer = non;
```

```
                                else
                                        {
                                        bloc.eliminer = oui;
                                        fichier.seekp((i - 1)*sizeof(blocs), ios::beg);
                                        fichier.write((char*) & bloc, sizeof(bloc));
                                        }
                                }
        }

/*-------------------------------------------------------------------------------------*/
/* SOUS-PROGRAMME : PRESEANCE_PARTIE2                                                  */
        */
/* DESCRIPTION :        Ce sous-programme est appelé par le programme principal        */
/* Il crée les arcs de précéance      entre les blocs. Éventuellement ces arcs         */
/* auront des capacités infinies lors de la construction du graph.                     */
/*-------------------------------------------------------------------------------------*/
void preseance_partie2(int nbr_bloc, int nbr_l, int nbr_c)
        {
                /* PARTIE DÉCLARATION */

        blocs bloc1, bloc2;

        int i, j;                       //les compteurs de blocs.
        int compteur_pred;              //compteur des prédécesseurs.

                /* PARTIE INSTRUCTIONS */

        for ( i = (nbr_l *nbr_c + 1); i <= nbr_bloc; i++)
                {
                initialisation(bloc1);
                fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
                fichier.read((char*) & bloc1, sizeof(bloc1));
                if (bloc1.eliminer == non)
                        {
                        compteur_pred = 0;                      //initialisation des variables
                        j = i - (nbr_l * nbr_c);
                        initialisation(bloc2);
                        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
                        fichier.read((char*) & bloc2, sizeof(bloc2));
                        if (bloc2.eliminer == non)//On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
                                {
                                bloc1.predecesseurs[compteur_pred] = j;
                                compteur_pred = compteur_pred + 1;
                                }

                        j = j + 1;
                        initialisation(bloc2);
                        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
                        fichier.read((char*) & bloc2, sizeof(bloc2));
                        if (bloc2.eliminer == non)//On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
```

```
                {
                bloc1.predecesseurs[compteur_pred] = j;
                compteur_pred = compteur_pred + 1;
                }

        j = j - 2;
        initialisation(bloc2);
        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
        fichier.read((char*) & bloc2, sizeof(bloc2));
        if (bloc2.eliminer == non) //On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
                {
                bloc1.predecesseurs[compteur_pred] = j;
                compteur_pred = compteur_pred + 1;
                }

        j = j - nbr_l;
        initialisation(bloc2);
        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
        fichier.read((char*) & bloc2, sizeof(bloc2));
        if (bloc2.eliminer == non) //On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
                {
                bloc1.predecesseurs[compteur_pred] = j;
                compteur_pred = compteur_pred + 1;
                }

        j++;
        initialisation(bloc2);
        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
        fichier.read((char*) & bloc2, sizeof(bloc2));
        if (bloc2.eliminer == non) //On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
                {
                bloc1.predecesseurs[compteur_pred] = j;
                compteur_pred = compteur_pred + 1;
                }

        j++;
        initialisation(bloc2);
        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
        fichier.read((char*) & bloc2, sizeof(bloc2));
        if (bloc2.eliminer == non) //On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
                {
                bloc1.predecesseurs[compteur_pred] = j;
                compteur_pred = compteur_pred + 1;
                }

        j = j + 2*nbr_l;
        initialisation(bloc2);
        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
```

```
                        fichier.read((char*) & bloc2, sizeof(bloc2));
                        if (bloc2.eliminer == non) //On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
                                {
                                bloc1.predecesseurs[compteur_pred] = j;
                                compteur_pred = compteur_pred + 1;
                                }

                        j--;
                        initialisation(bloc2);
                        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
                        fichier.read((char*) & bloc2, sizeof(bloc2));
                        if (bloc2.eliminer == non) //On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
                                {
                                bloc1.predecesseurs[compteur_pred] = j;
                                compteur_pred = compteur_pred + 1;
                                }

                        j--;
                        initialisation(bloc2);
                        fichier.seekg((j - 1)*sizeof(blocs), ios::beg);
                        fichier.read((char*) & bloc2, sizeof(bloc2));
                        if (bloc2.eliminer == non) //On vérifie si le bloc qu'on test comme prédecesseur
n'a pas déjà été éliminé
                                {
                                bloc1.predecesseurs[compteur_pred] = j;
                                compteur_pred = compteur_pred + 1;
                                }

                        if ( compteur_pred != 9)
                                bloc1.eliminer = oui;

                        fichier.seekp((i - 1)*sizeof(blocs), ios::beg);
                        fichier.write((char*) & bloc1, sizeof(bloc1));
                        }
                }
        }

/*-------------------------------------------------------------------------------------*/
/* SOUS-PROGRAMME : ECRITURE_FINPUT
        */
/* DESCRIPTION :        Ce sous-programme est appelé par le programme principal.        */
/* Il sert à écrire le fichier INPUT qui sera envoyé dans lors de la                    */
/* résolution du flow max.  Le fichier qui sera lut et envoyé pour la                   */
/* résolution du flow max devra avoir le format suivant:                                */
/* p Problème Noeuds Arcs       - Problème = max dans notre cas
        */
/* n no s/t                     - nom des noeuds source et puit.  Le noeud source aura le   */
/*                              - (nbr_noeuds + 1) et le noeud puit aura le (nbr_noeuds + 2)  */
/* a src dst cap                - information sur les arcs. L'origine, la destination et la  */
```

```
/*                                  - capacité de l'arc.
        */
/* Tout d'abord le programme devra calculer le nombre d'arcs qu'il faut pour bâtir le graph.      */
/* Les noeuds qui sont à la surface ne sont relié qu'au noeud source ou au noeud puits selon      */
/* qu'il est négatif ou positif. Quant aux autres noeuds ils ont 9 prédécesseurs et il sont  */
/* eux aussi reliés au noeud source ou au noeud puits.
        */
/* Il nous faudra donc la valeur du niveau minimum pour pouvoir déterminer quel sont les noeuds
        */
/* qui se trouvent à la surface du gisement.                                                     */
/*-------------------------------------------------------------------------------------------*/

void ecriture_finput (int nbr_l, int nbr_c, int nbr_bloc, int periode)
        {

                /* PARTIE DÉCLARATION */

        ofstream fichier_a_ecrire;   //ouverture du fichier en mode ecriture

        blocs bloc;

        int i, j;
  int compteur_waste, compteur_minerai, compteur_blocs;
        int nbr_arcs;

                /* INITIALISATION DES VARIABLES */

        nbr_arcs = 0;
        compteur_waste = 0;
        compteur_minerai = 0;
        compteur_blocs = 0;

                /* PARTIE INSTRUCTION */

        for ( i = (nbr_l*nbr_c + 1); i <= nbr_bloc; i++)
                {
                initialisation(bloc);
                fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
                fichier.read((char*) & bloc, sizeof(bloc));
                if ( bloc.eliminer == non)
                        nbr_arcs = (nbr_arcs + 10);
                }
        nbr_arcs = nbr_arcs + (nbr_l * nbr_c);

        fichier_a_ecrire.open("input.max");          //ouverture du fichier en mode écriture
        if ( fichier_a_ecrire.fail())
                cout << "(7) Problème d'ouverture du fichier input.max";
        else
                {
                fichier_a_ecrire << "p max "<< (nbr_bloc + 2)<<" "<< nbr_arcs << endl;
                fichier_a_ecrire << "n " << (nbr_bloc + 1) << " s \n";
                fichier_a_ecrire << "n " << (nbr_bloc + 2) << " t \n";
```

```
for (i = 1; i <= nbr_l*nbr_c; i++)
        {
        initialisation(bloc);
        fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
        fichier.read((char*) & bloc, sizeof(bloc));
        if ( bloc.eliminer == non)
                {
                compteur_blocs ++;
                if (bloc.cout[periode] > 0)
                        {
                        fichier_a_ecrire <<"a "<< (nbr_bloc + 1) <<" "<< i <<" "<<
bloc.cout[periode]<< endl;
                        compteur_minerai ++;
                        }
                else
                        {
                        fichier_a_ecrire <<"a "<< i <<" "<< (nbr_bloc + 2) <<" "<<-
(bloc.cout[periode])<< endl;
                        compteur_waste ++;
                        }
                }
        }

for (i = nbr_l*nbr_c + 1; i <= nbr_bloc; i++)
        {
        initialisation(bloc);

        fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
        fichier.read((char*) & bloc, sizeof(bloc));
        if ( bloc.eliminer == non)
                {
                compteur_blocs ++;
                for (j = 0; j < 9; j++)
                        {
                        fichier_a_ecrire <<"a "<< i <<" "<< bloc.predecesseurs[j];
                        fichier_a_ecrire << " " << infini << endl;
                        }

                if (bloc.cout[periode] > 0)
                        {
                        fichier_a_ecrire <<"a "<< (nbr_bloc + 1) <<" "<< i <<" "<<
bloc.cout[periode]<< endl;
                        compteur_minerai ++;
                        }
                else
                        {
                        fichier_a_ecrire <<"a "<< i <<" "<< (nbr_bloc + 2) <<" "<<-
(bloc.cout[periode])<< endl;
                        compteur_waste ++;
                        }
                }
```

```
                }
        cout << endl;

        cout <<"Il y a "<<compteur_blocs<<" blocs en tout dans le cone. "<< endl;
        cout <<"Il y a "<<compteur_waste<<" blocs de stérile et "<< endl;
        cout <<"Il y a "<<compteur_minerai<<" blocs de minerai. "<< endl;
        }
    fichier_a_ecrire.close();
    }

/*-------------------------------------------------------------------------*/
/* SOUS-PROGRAMME : RESULTATS_FLOTMAX
    */
/* DESCRIPTION :    Ce sous-programme est appelé par le programme principal    */
/* à l'intérieur de la boucle pour le calcul des périodes.                    */
/* RESULTATS_FLOTMAX sert à transformer les données du fichier coupe.out      */
/* fournit par le flot max et à mettre à jour les données dans le fichier  */
/* data.dat.                                                                  */
/*-------------------------------------------------------------------------*/
void resultats_flowmax (int periode)

    {

        /* PARTIE DÉCLARATION */

    blocs bloc;
    int noeud;

        /* INITIALISATION DES VARIABLES */

    noeud = 0;

        /* PARTIE INSTRUCTION */

    fichier_a_lire.open("coupe.out");
    if ( fichier_a_lire.fail())
        cout << "(9) Problème d'ouverture de fichier coupe.out";
    else
        {
        while (!fichier_a_lire.eof())
            {
            fichier_a_lire >> noeud;
            noeud;
            initialisation(bloc);
            fichier.seekg((noeud)*sizeof(blocs), ios::beg);
            fichier.read((char*) & bloc, sizeof(bloc));
            if (bloc.periode_extraction == 0)
                {
                bloc.periode_extraction = periode + 1;
                fichier.seekp((noeud)*sizeof(blocs), ios::beg);
                fichier.write((char*) & bloc, sizeof(bloc));
                }
```

```
                    }
                }
        fichier_a_lire.close();
        }

/*-------------------------------------------------------------------------  */
/* SOUS-PROGRAMME : ECRITURE_FOUTPUT                                          */
/* DESCRIPTION : Ce sous-programme est appelé par le programme principal.     */
/* Il sert à écrire le fichier OUTPUT qui sera envoyé dans le logiciel        */
/* SURPAC afin de visualiser les résultats de l'optimisation.                 */
/* Le fichier qui sera envoyé devra avoir le format suivant:                  */
/* X     Y       Z       EXTRAIT? ou période d'extraction                     */
/* Si le bloc n'est jamais extrait = 0                                        */
/* Si le bloc est extrait il aura une valeur positive > 0 selon la période    */
/* d'extraction.                                                              */
/*-------------------------------------------------------------------------  */
void ecriture_foutput (int nbr_bloc)

        {

                /* PARTIE DÉCLARATION */

        ofstream fichier_a_ecrire;          //ouverture du fichier en mode écriture

        blocs bloc;
        int i, string, rock_code, mcaf, pcaf;
        char fichier_ecriture[50];

                /* PARTIE INSTRUCTION */

        string = 1;

        cout << "Donnez-moi le nom de votre fichier des résultats : "<<endl;
        cin >> fichier_ecriture;
        fichier_a_ecrire.open(fichier_ecriture);      //ouverture du fichier en mode ecriture
        if ( fichier_a_ecrire.fail())
                cout << "(11) Problème d'ouverture de fichier output.txt";
        else
                {
                for ( i = 1; i <= nbr_bloc; i++)
                        {
                        initialisation(bloc);

                        fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
                        fichier.read((char*) & bloc, sizeof(bloc));
                        if ((bloc.litho == 99)||(bloc.litho == 19))
                                {
                                rock_code = 99;
                                mcaf = 0;
                                pcaf = 0;
                                }
                        if (bloc.litho == 103)
```

```
                                {
                                rock_code = 103;
                                mcaf = 1;
                                pcaf = 1;
                                }
                        if ((bloc.litho == 100)||(bloc.litho == 101)||(bloc.litho == 102)||(bloc.litho ==
104)
                        ||(bloc.litho == 106)||(bloc.litho == 108)||(bloc.litho == 202)||(bloc.litho ==
203))
                                {
                                rock_code = 100;
                                mcaf = 1;
                                pcaf = 0;
                                }

                        fichier_a_ecrire <<string<<" , "<<bloc.coor_y<<" , "<<bloc.coor_x;
                        fichier_a_ecrire <<" , "<<bloc.coor_z<<" , "<<bloc.litho<<" , ";
                        fichier_a_ecrire <<bloc.teneur<<" , "<<bloc.wrec<<" , "<<bloc.densite;
                        fichier_a_ecrire <<" , "<<rock_code<<" , "<<mcaf<<" , "<<pcaf;
                        fichier_a_ecrire  <<"  ,  "<<bloc.periode_extraction<<"  ,  "<<bloc.eliminer<<
endl;
                        }
                }
        fichier_a_ecrire.close();
        }


/*-------------------------------------------------------------------------  */
/*FICHIER : PARAM.CPP                                                          */
/*NOM :         Geneviève Auger                                                */
/*DESCRIPTION :         Ce programme détermine les contours optimaux d'une     */
/* exploitation minière pour différents prix de minerai.  On veut reproduire  */
/* les résultats de Whittle en faisant une paramétrisation du gisement en     */
/* fonction du prix.  Le programme crer le graphe de Picard et pour chacun    */
/* des prix va résoudre un flot max contrairement à Whittle qui résout le     */
/* problème des contours ultimes à l'aide de L&G.                             */
/* Le problème flow max pour un prix donné est résolu par l'algorithme de     */
/* Goldberg.                                                                  */
/*-------------------------------------------------------------------------  */

int main(void)
        {

                /* DÉCLARATION DES PROTOTYPES DES FONCTIONS */

        void initialisation(blocs & bloc);

        void lecture_ligne(float& coor_y, float& coor_x, float& coor_z, float& lito,
                        float& ten, float& wr, float& dens);

        void nouvelle_borne(int& min, int& max, int inferieur, int superieur, int nombre, int pas);
```

```
        void oui_non(int& reponse);

        void donnee_reduction(int& inf_x, int& sup_x, int& inf_y, int& sup_y,
                        int& inf_z, int& sup_z, int& nbr_niveau, int& nbr_ligne,
                        int& nbr_colonne, int& nbr_blocs, int prof, int haut, int larg);

        void lecture(type_nom fichier_lecture, int& nbr_bloc, int& nbr_n, int& nbr_l,int& nbr_c,
                        int& l_min, int& l_max, int& c_min, int& c_max, int& n_min, int& n_max,
                        int& prof, int& larg, int& haut);

        void cas_1_1(type_nom fichier_lecture, int& nbr_bloc, int& nbr_n, int& nbr_l, int& nbr_c,
                        int& l_min, int& superieur_x, int& inferieur_y, int& superieur_y,
                        int& inferieur_z, int& superieur_z, int prof, int larg, int haut, int reference);

        void cas_1_2(type_nom fichier_lecture, int prof, int haut, int larg, int l_min,
                        int l_max, int c_min, int c_max, int n_min, int n_max, int nbr_l,
                        int nbr_c, int nbr_n, int nbr_bloc);

        void cas_2_1(type_nom fichier_lecture, int nbr_l, int nbr_c,
                        int l_min, int inferieur_y, int inferieur_z, int superieur_z,
                        int prof, int larg, int haut, int reference);

        void calcul_cout(int& nbr_per, int nbr_bloc, int larg, int haut, int prof);

        void preseance_partie1(int nbr_n, int nbr_l, int nbr_c);

        void preseance_partie2(int nbr_bloc, int nbr_l, int nbr_c);

        void ecriture_finput(int nbr_l, int nbr_c, int nbr_bloc, int periode);

        void ecriture_foutput(int nbr_bloc);

        void resultats_flowmax(int periode);

                /* PARTIE DÉCLARATION DES VARIABLES */

        int nombre_bloc, nombre_periode, nombre_niveau, nombre_ligne, nombre_colonne;
        int niveau_minimum, niveau_maximum, ligne_minimum, ligne_maximum, colonne_minimum,
colonne_maximum;
        int profondeur, largeur, hauteur;
        int p, optimise, systeme_reference, reduire;
        int bidon;

                /* INITIALISATION DES VARIABLES*/

        nombre_bloc = 0;
        nombre_periode = 0;
        nombre_niveau = 0;
        niveau_minimum = 0;
        nombre_ligne = 0;
        nombre_colonne = 0;
        niveau_minimum = 0;
```

```
niveau_maximum = 0;
ligne_minimum = 0;
ligne_maximum = 0;
colonne_minimum = 0;
colonne_maximum = 0;
profondeur = 0;
largeur = 0;
hauteur = 0;
optimise = 2;
systeme_reference = 0;

        /* PARTIE INSTRUCTIONS */

fichier.open("data.dat", ios::binary|ios::in|ios::out);
if ( fichier.fail())
        cout << "(4) Problème d'ouverture du fichier data.dat"<<endl;
else
        {
        cout << "Entrez le nom du fichier de donnees de surpac : ";
        cin >> nom_fichier;

        lecture(nom_fichier, nombre_bloc, nombre_niveau, nombre_ligne, nombre_colonne,
                ligne_minimum, ligne_maximum, colonne_minimum, colonne_maximum,
niveau_minimum,
                niveau_maximum, profondeur, largeur, hauteur);

        cout << "Desirez-vous reduire ce gisement?"<<endl;
        oui_non(reduire);

        cout << endl << "Désirez-vous poursuivre avec l'optimisation immédiatement? " <<endl;
        oui_non(optimise);

        if (optimise == 1)
                {
                cout << endl << "L'axe des z du systeme de reference de votre gisement est-il
vers le haut?"<<endl;
                oui_non(systeme_reference);
                }

        if ((reduire == 1) && (optimise == 1))
                {
                cas_1_1(nom_fichier, nombre_bloc, nombre_niveau, nombre_ligne,
nombre_colonne,
                ligne_minimum, ligne_maximum, colonne_minimum, colonne_maximum,
niveau_minimum,
    niveau_maximum, profondeur, largeur, hauteur, systeme_reference);
                }
        if ((reduire == 1) && (optimise == 2))
                {
                cas_1_2(nom_fichier, profondeur, hauteur, largeur, ligne_minimum,
                ligne_maximum, colonne_minimum, colonne_maximum, niveau_minimum,
```

```
                    niveau_maximum,    nombre_ligne,    nombre_colonne,    nombre_niveau,
nombre_bloc);
                    }
            if ((reduire == 2) && (optimise == 1))
                    {
                    cas_2_1(nom_fichier,    nombre_ligne,    nombre_colonne,    ligne_minimum,
colonne_minimum,
                            niveau_minimum, niveau_maximum, profondeur, largeur, hauteur,
                            systeme_reference);
                    }
            if ((reduire == 2) && (optimise == 2))
                    {
                    cout << "Il n'y a rien à faire....." << endl ;
                    }

            if (systeme_reference == 1)
                    {
                    bidon = nombre_colonne;
                    nombre_colonne = nombre_ligne;
                    nombre_ligne = bidon;
                    }

            if (optimise == 1)
                    {
                    preseance_partie1(nombre_niveau, nombre_ligne, nombre_colonne);
                    preseance_partie2(nombre_bloc, nombre_ligne, nombre_colonne);
                    calcul_cout(nombre_periode, nombre_bloc, largeur, hauteur, profondeur);
                    for ( p = 0; p < nombre_periode; p++)
                            {
                            ecriture_finput (nombre_ligne, nombre_colonne, nombre_bloc, p);
                            system("gold_q input.max /dev/null");
                            resultats_flowmax(p);
                            }
                    ecriture_foutput(nombre_bloc);
                    }
            }
    fichier.close();
    }
```

A3.2 «Param_inv.cpp»

En deuxième partie, on présente le «listing» des modification apportées au programme original «Param.cpp» pour permettre d'inverser le graphe. Il s'agit d'un programme qui calcule des coûts pour chacun des blocs à partir des teneurs et pour un prix du métal donné par l'usager, mais qui, au contraire du programme original, créera le graphe de Picard, pour le flot maximum, en reliant le nœud puits à tous les nœuds qui ont une

valeur négative et en reliant tous les nœuds qui ont une valeur positive au nœud source. Il s'agit du programme nommé «Param_inv.cpp» dans le mémoire.

«Listing» de la procédure qui a été modifiée par rapport au programme original :

```
/*-------------------------------------------------------------------------------------*/
/* SOUS-PROGRAMME : ECRITURE_FINPUT                                                    */
/* DESCRIPTION :        Ce sous-programme est appelé par le programme principal.*/
/* Il sert à écrire le fichier INPUT qui sera envoyé dans lors de la       */
/* résolution du flow max.  Le fichier qui sera lut et envoyé pour la */
/* résolution du flow max devra avoir le format suivant:       */
/* p Problème Noeuds Arcs          - Problème = max dans notre cas    */
/* n no s/t                        - nom des noeuds source et puit.  Le noeud source aura le */
/*                                 - (nbr_noeuds + 1) et le noeud puit aura le (nbr_noeuds + 2)   */
/* a src dst cap                   - information sur les arcs. */
/*                                 L'origine, la destination et la capacité de l'arc.*/
/* Tout d'abord le programme devra calculer le nombre d'arcs qu'il faut pour */
/* bâtir le graph.  Les noeuds qui sont à la surface ne sont relié qu'au noeud */
/* source ou au noeud puits selon qu'il est négatif ou positif.  Quant aux */
/* autres noeuds ils ont 9 prédécesseurs et il sont eux aussi reliés au noeud */
/* source ou au noeud puits. Il nous faudra donc la valeur du niveau minimum */
/* pour pouvoir déterminer quel sont les noeuds       qui se trouvent à la surface */
/* du gisement. Cette fonction diffère du programme original param_niv.cpp */
/* parce qu'il inverse tous les arcs du graphe original.                               */
/*-------------------------------------------------------------------------------------*/

void ecriture_finput (int nbr_l, int nbr_c, int nbr_bloc, int periode)
        {

                /* PARTIE DÉCLARATION */

        ofstream fichier_a_ecrire;   //ouverture du fichier en mode ecriture

        blocs bloc;

        int i, j, compteur_waste, compteur_minerai, compteur_blocs;
        int nbr_arcs;

                /* INITIALISATION DES VARIABLES */

        nbr_arcs = 0;
        compteur_waste = 0;
        compteur_minerai = 0;
        compteur_blocs = 0;

                /* PARTIE INSTRUCTION */

        for ( i = (nbr_l*nbr_c + 1); i <= nbr_bloc; i++)
                {
```

```
                initialisation(bloc);
                fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
                fichier.read((char*) & bloc, sizeof(bloc));
                if ( bloc.eliminer == non)
                        nbr_arcs = (nbr_arcs + 10);
                }
        nbr_arcs = nbr_arcs + (nbr_l * nbr_c);

        fichier_a_ecrire.open("input.max");          //ouverture du fichier en mode écriture
        if ( fichier_a_ecrire.fail())
                cout << "(7) Problème d'ouverture du fichier input.max";
        else
                {
                fichier_a_ecrire << "p max "<< (nbr_bloc + 2)<<" "<< nbr_arcs << endl;
                fichier_a_ecrire << "n " << (nbr_bloc + 1) << " s \n";
                fichier_a_ecrire << "n " << (nbr_bloc + 2) << " t \n";

                for (i = 1; i <= nbr_l*nbr_c; i++)
                        {
                        initialisation(bloc);
                        fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
                        fichier.read((char*) & bloc, sizeof(bloc));
                        if ( bloc.eliminer == non)
                                {
                                compteur_blocs ++;
                                if (bloc.cout[periode] > 0)
                                        {
                                        fichier_a_ecrire <<"a "<< i <<" "<<(nbr_bloc + 1)<<" "<<
bloc.cout[periode]<< endl;
                                        compteur_minerai ++;
                                        }
                                else
                                        {
                                        fichier_a_ecrire <<"a "<<(nbr_bloc + 2)<<" "<<i<<" "<<-
(bloc.cout[periode])<< endl;
                                        compteur_waste ++;
                                        }
                                }
                        }

                for (i = nbr_l*nbr_c + 1; i <= nbr_bloc; i++)
                        {
                        initialisation(bloc);

                        fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
                        fichier.read((char*) & bloc, sizeof(bloc));
                        if ( bloc.eliminer == non)
                                {
                                compteur_blocs ++;
                                for (j = 0; j < 9; j++)
                                        {
                                        fichier_a_ecrire <<"a "<< bloc.predecesseurs[j]<<" "<<i;
```

```
                                        fichier_a_ecrire << " " << infini << endl;
                                        }

                                if (bloc.cout[periode] > 0)
                                        {
                                        fichier_a_ecrire <<"a "<<i<<" "<< (nbr_bloc + 1) <<" "<<
bloc.cout[periode]<< endl;
                                        compteur_minerai ++;
                                        }
                                else
                                        {
                                        fichier_a_ecrire <<"a "<<(nbr_bloc + 2)<<" "<<i<<" "<<-
(bloc.cout[periode])<< endl;
                                        compteur_waste ++;
                                        }
                                }
                        }
                cout << endl;

                cout <<"Il y a "<<compteur_blocs<<" blocs en tout dans le cone. "<< endl;
                cout <<"Il y a "<<compteur_waste<<" blocs de stérile et "<< endl;
                cout <<"Il y a "<<compteur_minerai<<" blocs de minerai. "<< endl;
                }
        fichier_a_ecrire.close();
        }
```

A3.3 «Random.cpp» ou «Random_ninv.cpp»

En troisième partie, on présente le code des modifications qui ont été apportées au programme original «Param.cpp» pour permettre d'attribuer des coûts de façon aléatoire. Il s'agit d'un programme qui calcule des coûts pour chacun des blocs à partir d'une fonction aléatoire et qui attribue au bloc une certaine valeur qui doit se situer à l'intérieur d'un intervalle prédéterminé. Nous avons éliminé les valeurs qui représentent la litho, la teneur, la densité et le «wrec» afin de sauver un peu d'espace mémoire. C'est pour cette raison que la majorité des procédures qui utilisaient ces données seront modifiées légèrement. Par exemple, la procédure de lecture du fichier et toutes les procédures qui l'appellent. Toutefois, la modification majeure se trouve dans la procédure qui calcule les coûts. Nous n'avons pas mis toutes les modifications qui ont été faites dans ce programme, mais seulement les plus importantes, soit la structure de l'information, la procédure de lecture d'une ligne et la procédure qui calcule les coûts. Dans le mémoire, il s'agit du programme nommé «Random.cpp» ou «Random_ninv.cpp».

«listing» des modifications qui ont été apportées par rapport au programme original :

```
#include <stdlib.h>                    //Pour l'utilsation de system()
#include <time.h>

#define max_periodes 50                //bon pour un maximum de une fosse
#define non 0
#define oui 1
#define infini 100000000

struct blocs
        {
        int coor_x;
        int coor_y;
        int coor_z;
        int cout[max_periodes];
        int predecesseurs[9];
        int eliminer;
        int periode_extraction;
        };


typedef char type_nom[50];

type_nom nom_fichier;
fstream fichier;

ifstream fichier_a_lire;

/*-------------------------------------------------------------------------*/
/* SOUS-PROGRAMME : INITIALISATION                                         */
/* DESCRIPTION : Ce programme est une sous-routine qui est appelée chaque*/
/* fois qu'une variable de type blocs doit être initialisée                */
/*-------------------------------------------------------------------------*/
void initialisation(blocs & bloc)
                //Le paramètre est transmis par adresse dans cette fonction.
        {
                /* PARTIE DÉCLARATION */
        int i;

                /*PARTIE INSTRUCTIONS */

        bloc.coor_x = 0;
        bloc.coor_y = 0;
        bloc.coor_z = 0;
        for (i = 0; i < max_periodes; i++)
                bloc.cout[i] = 0;
        for (i = 0; i < 9; i++)
                bloc.predecesseurs[i] = 0;
        bloc.eliminer = non;
        bloc.periode_extraction = 0;
        }
```

```
/*-------------------------------------------------------------------------- */
/* SOUS-PROGRAMME : LECTURE_LIGNE*/
/* DESCRIPTION : Ce programme est une sous-routine qui permet de lire une ligne*/
/* du fichier initial de Surpac. Le fichier d'input doit se présenter sous la */
/* forme suivante: String, y, x, z, litho, teneur, wrec, densite, rock_code, mcaf, */
/* pcaf. Le fichier ne doit comprendre aucune en-tête, ni de message pour */
/* indiquer la fin du fichier.*/
/*-------------------------------------------------------------------------- */
void lecture_ligne(float& coor_y, float& coor_x, float& coor_z)

        //Les paramètres sont transmis par adresse dans cette fonction.
{

        /* PARTIE DÉCLARATION DES VARIABLES */

float lito, ten, wr, dens, r_c, mf, pf;
char e;
int string;

        /* PARTIE INITIALISATION DES VARIABLES */

ten = 0;
wr = 0;
dens = 0;
string = 0;
coor_y = 0;
coor_x = 0;
coor_z = 0;
lito = 0;
r_c = 0;
mf = 0;
pf = 0;

        /* PARTIE INSTRUCTION */

fichier_a_lire >> string;
fichier_a_lire >> e;
fichier_a_lire >> coor_y;
fichier_a_lire >> e;
fichier_a_lire >> coor_x;
fichier_a_lire >> e;
fichier_a_lire >> coor_z;
fichier_a_lire >> e;
fichier_a_lire >> lito;
fichier_a_lire >> e;
fichier_a_lire >> ten;
fichier_a_lire >> e;
fichier_a_lire >> wr;
fichier_a_lire >> e;
fichier_a_lire >> dens;
fichier_a_lire >> e;
```

```
        fichier_a_lire >> r_c;
        fichier_a_lire >> e;
        fichier_a_lire >> mf;
        fichier_a_lire >> e;
        fichier_a_lire >> pf;
        }
/*-------------------------------------------------------------------------------- */
/* SOUS-PROGRAMME : CALCUL_COUT                                                    */
/* DESCRIPTION :      Ce sous-programme est appelé par le programme principal      */
/* Il calcul les couts pour chacun des blocs et pour toutes les périodes. */
/* Il demande tout d'abord le prix du métal, le nombre de périodes voulue (- de 50) */
/* puis il calcul des couts pour chacune des périodes.                             */
/*-------------------------------------------------------------------------------- */
void calcul_cout(int& nbr_per, int nbr_bloc)

        {
                /* PARTIE DÉCLARATION */

        blocs bloc;

        int i, j;
        int pourcentage_sterile, minerai_sterile;
        float coupe_source, coupe_puit;

                /* PARTIE INSTRUCTIONS */

        srand(time(NULL));

        coupe_source = 0;
        coupe_puit = 0;

        cout << "Combien de periodes voulez-vous? (maximum de 50) : ";
        cin >> nbr_per;
        cout << endl;
        while ((nbr_per < 1) && (nbr_per > 50))
                {
                cout << "Recommencez s.v.p. , (maximum 50 periodes) : ";
                cin >> nbr_per;
                cout << endl;
                }

        cout << "Combien de blocs de stérile sur 100 voulez-vous?(un chiffre entre 1 et 100):";
        cin >> pourcentage_sterile;
        cout << endl;
        while ((pourcentage_sterile < 1) && (pourcentage_sterile > 100))
                {
                cout << "Recommencez s.v.p. , (un chiffre entre 1 et 100) : ";
                cin >> pourcentage_sterile;
                cout << endl;
                }
```

```
for ( i = 1; i <= nbr_bloc; i++)
        {
        initialisation(bloc);
        fichier.seekg((i - 1)*sizeof(blocs), ios::beg);
        fichier.read((char*) & bloc, sizeof(bloc));
        if (bloc.eliminer == non)
                {
                for (j = 0; j < nbr_per; j++)
                        {
                        minerai_sterile = rand() % 100;
                        if (minerai_sterile >= pourcentage_sterile)
                                {
                                bloc.cout[j] = 6050 + (rand() % 20000)*4;
                                coupe_source = coupe_source + bloc.cout[j];
                                }
                        else
                                {
                                bloc.cout[j] = - 8000;
                                coupe_puit = coupe_puit + bloc.cout[j];
                                }
                        }
                }
        fichier.seekp((i - 1)*sizeof(blocs), ios::beg);
        fichier.write((char*) & bloc, sizeof(bloc));
        }
cout << "La coupe à la source est de : "<<coupe_source<< endl;
cout << "La coupe au puit est de : "<<coupe_puit<< endl;
}
```

A3.4 «Random_inv.cpp»

Le dernier programme qui a été fait pour réaliser les tests permet d'attribuer des coûts de façon aléatoire et de plus, d'inverser le graphe. En fait, nous ne présenterons pas le code de ce programme étant donné qu'il s'agit d'une combinaison des programmes «Random.cpp» et «Param_inv.cpp». Le code est le même que Random.cpp à l'exception de la procédure qui écrit le fichier et qui sera envoyée au flot maximum. Cette dernière est identique à celle qui a été présentée en A3.2

ANNEXE IV
RÉSULTATS

Tableau A4. 1 : Résultats des tests pour le fichier no 1 : 1398011

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué	diff. 100000
1	1398011	139	80	11	122320	99770	oui	230

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	04/08/2000 07/08/2000	paramninv.cpp id	32 id	92130 id	7640 id	10962 id	8.42 8.4 8.3 8.1	8.31
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	17/08/2000 30/08/2000	random.cpp id	random id	69922 69780 69937 69968 70130 69871 70112 69942 69599 69952	29848 29990 29833 29802 26640 29899 29658 29828 30171 29818	93497 93806 93889 93823 93726 93721 93826 93810 93864 93605	25.88 43.71 25.87 25.48 32.99 34.21 27.6 26.77 25.28 32.96	30.08

Tableau A4. 2 : Résultats des tests pour le fichier no 2 : 1257116

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué	diff. 100000
2	1257116	125	71	16	142000	99920	oui	80

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	04/08/2000	paramninv.cpp	32	?	?	19845	13.8	
	07/08/2000	id	id	id	id	id	13.99	
							14.06	
							13.79	
								13.91
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	17/08/2000	random.cpp	random	69938	29982	96582	31.3	
				69824	30096	96246	34.41	
				69983	29937	96634	34.06	
	21/08/2000	id	id	69967	29953	96282	34.51	
				69812	30108	96414	31.48	
	30/08/2000	id	id	70031	29889	96312	41.98	
				70071	29849	96331	31.57	
				69902	30018	96573	35.07	
				70206	29714	96529	32.06	
				69984	29936	96327	34.72	
								34.12

Tableau A4. 3 : Résultats des tests pour le fichier no 3 : 1306221

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué	diff. 100000
3	1306221	130	62	21	169260	100100	oui	-100

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
***Test pour 1 periode avec Copernic	04/08/2000	paramninv.cpp	32	?	?	30035	25.42	25.42
Test pour 1 periode avec Ptolemee	07/08/2000	paramninv.cpp	32	?	?	30035	19.4 19.56 19.54	19.50
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	17/08/2000	random.cpp	random	69911	30189	98174	35.26	
				69920	30180	98134	37.26	
	21/08/2000	id	id	69700	30400	98225	32.51	
				70275	29825	98176	35.37	
				70053	30047	98191	42.63	
	30/08/2000	id	id	70043	30057	98210	32.81	
				69990	30110	98189	41.33	
				69745	30355	98331	49.92	
				70257	29843	98041	44.1	
				70042	30058	98233	42.24	39.34

Tableau A4. 4 : Résultats des tests pour le fichier no 4 : 1385726

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué	diff. 100000
4	1385726	138	57	26	204516	99866	oui	134

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	04/08/2000	paramninv.cpp	32	?	?	42440	27.44	
	07/08/2000	id	id	id	id	id	26.49	
							26.56	
							26.59	
								26.77
***Test pour 1 periode avec Copernic	07/08/2000	paramninv.cpp	32	?	?	42440	33.88	
								33.88
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	17/08/2000	random.cpp	random	69879	29987	98906	46.88	
				70074	29792	99016	37.22	
	21/08/2000	id	id	69736	30130	99121	49.77	
				69959	29907	98911	44.27	
				70082	29784	99000	42.06	
	30/08/2000	id	id	69610	30256	98997	33.95	
				69924	29942	98909	44.78	
	31/08/2000	id	id	69931	29935	98991	42.71	
				69646	30220	99020	40.11	
				70061	29805	99046	40.03	
								42.18

Tableau A4. 5 : Résultats des tests pour le fichier no 5 : 1276031

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué	diff. 100000
5	1276031	127	60	31	236220	100130	non	-130

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	07/08/2000	paramninv.cpp	32	?	?	48367	41.97 42.36 41.76	42.03
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	17/08/2000	random.cpp	random	69984	30146	99449	45.13	
				70247	29883	99445	53.54	
	21/08/2000	id	id	70283	29892	99550	53.73	
				69868	30262	99701	58.34	
				70086	30044	99692	45.79	
	31/08/2000	id	id	70203	29927	99225	50.48	
				69976	30154	99602	44.79	
				70190	29940	99464	45.26	
				70186	29944	99395	45.53	
				70147	29983	99363	37.59	48.02

Tableau A4. 6 : Résultats des tests pour le fichier no 6 : 1046936

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué	diff. 100000
6	1046936	104	69	36	258336	100030	non	-30

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	07/08/2000	paramninv.cpp	32	?	?	63498	48.08 48.13 48.26	48.16
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	17/08/2000	random.cpp	random	70045	29985	99685	43.07	
				70036	29994	99671	42.89	
	21/08/2000	id	id	69985	30045	99691	51.31	
				69954	30076	99784	53.95	
	31/08/2000	id	id	70147	29883	99559	66.27	
				70130	29900	99574	50.93	
				69870	30160	99447	42.68	
				70001	30029	99813	61.59	
				69805	30225	99416	45.76	
				69991	30039	99437	42.53	50.10

Tableau A4. 7 : Résultats des tests pour le fichier no 7 : 505014

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué
7	505014	50	50	14	35000	20076	non

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	07/08/2000	paramninv.cpp	32	16931	3145	5183	1.93 1.93 1.96	1.94
Test pour 1 periode avec Ptolemee	15/08/2000	paraminv.cpp	32	16931	3145	14895	1.2	1.20
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	22/08/2000	random.cpp	random	14033 14018 13993 14030 14084 14035 14014 14029 14119 14098	6043 6058 6083 6046 5992 6041 6062 6047 5957 5978	19313 19280 19161 19293 19320 19443 19529 19376 19392 19381	6.14 5.8 6.36 5.74 6.37 7.74 4.93 4.39 6.37 5.75	5.96

Tableau A4. 8 : Résultats des tests pour le fichier no 8 pour le graphe non inversé

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué
8	888814	65	66	14	60060	39494	oui

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	07/08/2000	paraminv.cpp	32	?	?	7163	?	?	4.48	
							?	?	4.47	
							?	?	4.47	
										4.47
Test pour 1 période avec Ptolemee, avec un ratio d'environ 10% stérile et 90% minerai	23/08/2000	random.cpp	random	4028	35466	39348	14.373	-0.322	3.74	
				3886	35608	39351	14.351	-0.311	3.77	
				4030	35464	39316	14.332	-0.322	3.93	
	31/08/2000	id	id	3925	35569	39343	14.427	-0.314	4.04	
				39303	3996	35498	14.283	-0.320	4.03	
				39313	3988	35506	14.333	-0.319	3.99	
				39343	3903	35591	14.373	-0.312	3.99	
				39331	3992	35502	14.332	-0.319	3.96	
				39358	3986	35508	14.379	-0.319	4.12	
				39341	3995	35499	14.333	-0.320	4.05	
										3.96
Test pour 1 période avec Ptolemee, avec un ratio d'environ 20% stérile et 80% minerai	23/08/2000	random.cpp	random	7878	31616	39200	12.821	-0.630	3.83	
				7959	31535	39198	12.775	-0.637	3.84	
				7902	31592	39172	12.752	-0.632	4.94	
	05/09/2000	id	id	7877	31617	39179	12.806	-0.630	6.69	
				7956	31538	39186	12.757	-0.636	6.72	
				7923	31571	39166	12.771	-0.634	6.59	
				7814	31680	39199	12.754	-0.625	3.77	
				8035	31459	39184	12.717	-0.643	6.75	
				8012	31482	39187	12.684	-0.641	6.73	
				8164	31330	39149	12.617	-0.653	5.10	
										5.50
Test pour 1 période avec Ptolemee, avec un ratio d'environ 30% stérile et 70% minerai	23/08/2000	random.cpp	random	11784	27710	39031	11.205	-0.943	6.64	
				11794	27700	39053	11.296	-0.944	6.63	
				11991	27503	38998	11.140	-0.959	4.63	
	05/09/2000	id	id	11878	27616	39025	11.136	-0.950	6.81	
				11851	27643	39004	11.162	-0.948	6.88	
				11907	27587	39032	11.081	-0.953	6.88	
				11881	27613	39030	11.229	-0.950	6.85	
				11674	27820	39013	11.313	-0.934	6.80	
				11836	27658	39027	11.166	-0.947	6.81	
				11778	27716	39009	11.215	-0.942	6.72	
										6.57

Tableau A4. 8 : Résultats des tests pour le fichier no 8 pour le graphe non inversé (suite)

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 période avec Ptolemee, avec un ratio d'environ 40% stérile et 60% minerai	23/08/2000	random.cpp	random	15886	23608	38831	9.472	-1.270	6.63	
				15876	23618	38890	9.528	-1.270	6.56	
				15770	23724	38871	9.535	-1.262	6.64	
	05/09/2000	id	id	15831	23663	38883	9.608	-1.266	6.75	
				15856	23638	38888	9.536	-1.268	6.75	
				15713	23781	38829	9.583	-1.257	6.80	
				15893	23601	38866	9.547	-1.271	6.84	
				15664	23830	38870	9.609	-1.253	6.85	
				15924	23570	38835	9.506	-1.274	6.81	
				15791	23703	38858	9.551	-1.263	6.83	6.75
Test pour 1 période avec Ptolemee, avec un ratio d'environ 50% stérile et 50% minerai	23/08/2000	random.cpp	random	19790	19704	38719	7.981	-1.583	6.71	
				19623	19871	38622	8.045	-1.570	9.32	
				19747	19747	38681	7.967	-1.580	6.46	
	05/09/2000	id	id	19766	19728	38691	7.981	-1.581	9.64	
				19900	19594	38713	7.862	-1.592	7.63	
				19765	19729	38711	7.929	-1.558	7.86	
				19911	19583	38698	7.924	-1.593	7.62	
				19603	19891	38665	8.007	-1.568	7.53	
				19570	19924	38668	8.078	-1.566	9.65	
				19929	19565	38651	7.977	-1.594	9.71	8.21
Test pour 1 période avec Ptolemee, avec un ratio d'environ 60% stérile et 40% minerai	23/08/2000	random.cpp	random	23875	15619	38401	6.270	-1.910	9.06	
				23799	15695	38456	6.334	-1.904	10.08	
				23618	15876	38417	6.451	-1.889	9.12	
	05/09/2000	id	id	23677	15817	38456	6.372	-1.894	13.40	
				23946	15548	38366	6.292	-1.916	13.37	
				23854	15640	38317	6.303	-1.908	9.27	
				23711	15783	38449	6.352	-1.897	10.22	
				23581	15913	38429	6.425	-1.886	7.36	
				23707	15787	38519	6.452	-1.897	7.52	
				23647	15847	38380	6.406	-1.892	9.15	9.86
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	22/08/2000	random.cpp	random	27705	11789	38027	?	?	12.97	
				27788	11706	37975	?	?	14.55	
				27751	11743	37858	?	?	11.87	
				27633	11861	38005	?	?	11.77	
				27627	11867	37998	?	?	15.76	
				27708	11786	37947	?	?	13.17	
				27694	11800	38020	?	?	11.82	
				27686	11808	38075	?	?	10.09	
				27639	11855	38123	?	?	10.11	
				27840	11654	37972	?	?	11.77	
	23/08/2000	id	id	27547	11949	38063	4.849	-2.204	12.27	12.38

Tableau A4. 8 : Résultats des tests pour le fichier no 8 pour le graphe non inversé (suite)

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 période avec Ptolemee, avec un ratio d'environ 75% stérile et 25% minerai	06/09/2000	random.cpp	random	29834	9660	36244	3.884	-2.387	18.64	
				29542	9952	36172	4.003	-2.363	12.63	
				29484	10010	36520	4.064	-2.359	12.49	
				29730	9764	36752	3.963	-2.378	12.70	
				29496	9998	36400	4.077	-2.360	12.59	
				29652	9842	36386	4.011	-2.372	18.43	
				29844	9650	36632	3.874	-2.387	17.40	
				29781	9713	36646	3.915	-2.382	17.20	
				29586	9908	36616	3.968	-2.367	12.49	
				29569	9925	36751	3.974	-2.365	15.33	14.99
Test pour 1 période avec Ptolemee, avec un ratio d'environ 80% stérile et 20% minerai	23/08/2000	random.cpp	random	31673	7821	33632	3.154	-2.530	16.79	
				31714	7780	33284	3.133	-2.537	17.08	
				31708	7786	32319	3.125	-2.537	19.97	
				31675	7819	31517	3.187	-2.534	17.29	
				31392	8102	34638	3.297	-2.511	19.33	
	06/09/2000	id	id	31558	7936	33445	3.224	-2.525	16.32	
				31490	8004	33718	3.196	-2.519	16.38	
				31564	7930	33150	3.219	-2.525	15.88	
				31710	7784	31848	3.123	-2.537	15.47	
				31680	7814	31344	3.137	-2.534	16.08	17.06
Test pour 1 période avec Ptolemee, avec un ratio d'environ 81% stérile et 19% minerai	06/09/2000	random.cpp	random	31916	7578	30905	3.046	-2.553	18.29	
				31961	7533	30519	3.041	-2.557	21.47	
				31876	7618	31484	3.083	-2.550	17.95	
				32045	7449	27987	2.995	-2.564	16.92	
	11/09/2000	random.cpp	random	31985	7509	31905	3.037	-2.559	18.90	
				31884	7610	31341	3.100	-2.551	19.06	
				32038	7456	28699	3.010	-2.563	16.28	
				31948	7546	28198	3.044	-2.556	17.22	
				32043	7451	30537	3.033	-2.563	22.10	
				31992	7502	30953	3.030	-2.559	19.19	18.74
Test pour 1 période avec Ptolemee, avec un ratio d'environ 82% stérile et 18% minerai	06/09/2000	random.cpp	random	32441	7053	22068	2.860	-2.595	15.60	
	11/09/2000	random.cpp	random	32363	7131	25068	2.898	-2.589	19.28	
				32516	6978	19541	2.807	-2.601	14.07	
				32357	7137	27655	2.888	-2.589	16.24	
				32332	7162	20638	2.861	-2.587	16.19	
				32490	7004	22274	2.832	-2.599	14.00	
				32453	7041	23224	2.824	-2.596	14.45	
				32381	7113	20993	2.856	-2.590	19.49	
				32432	7062	20741	2.856	-2.595	14.04	
				32222	7272	28647	2.938	-2.578	19.85	16.32

Tableau A4. 8 : Résultats des tests pour le fichier no 8 pour le graphe non inversé (suite)

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 période avec Ptolemee, avec un ratio d'environ 83% stérile et 17% minerai	23/08/2000	random.cpp	random	32777	6717	8557	2.735	-2.622	8.28	
				32688	6806	16100	2.758	-2.615	13.33	
				32818	6676	8873	2.706	-2.625	9.68	
	11/09/2000	random.cpp	random	32834	6660	14950	2.672	-2.627	11.60	
				32888	6606	10249	2.658	-2.631	11.03	
				32913	6581	5791	2.648	-2.633	7.66	
				32767	6727	19221	2.719	-2.621	16.99	
				32748	6746	12160	2.745	-2.620	10.54	
				32781	6713	15912	2.734	-2.622	13.99	
				32725	6769	14929	2.687	-2.618	11.25	
										11.44
Test pour 1 période avec Ptolemee, avec un ratio d'environ 84% stérile et 16% minerai	06/09/2000	random.cpp	random	33219	6275	6276	2.528	-2.657	5.12	
	11/09/2000	random.cpp	random	33086	6408	10769	2.600	-2.647	8.20	
				33163	6331	5999	2.572	-2.653	5.69	
				33219	6275	6467	2.548	-2.657	5.66	
				33223	6271	6187	2.532	-2.658	8.15	
				33161	6333	5851	2.581	-2.653	8.12	
				33243	6251	5791	2.511	-2.659	5.37	
				33061	6433	9239	2.627	-2.645	8.03	
				33300	6194	6065	2.511	-2.664	7.31	
				33278	6216	5352	2.488	-2.662	5.42	
										6.71
Test pour 1 période avec Ptolemee, avec un ratio d'environ 85% stérile et 15% minerai	23/08/2000	random.cpp	random	33541	5953	4006	2.402	-2.683	5.46	
				33717	5777	3808	2.337	-2.697	5.30	
				33557	5937	6903	2.428	-2.685	5.57	
	11/09/2000	random.cpp	random	33573	5921	4165	2.412	-2.686	5.08	
				33608	5886	4954	2.386	-2.689	4.96	
				33564	5930	5153	2.400	-2.685	5.12	
	12/09/2000	random.cpp	random	33654	5840	4348	2.358	-2.692	5.22	
				33543	5951	5229	2.390	-2.683	5.33	
				33515	5979	5191	2.408	-2.681	5.25	
				33612	5882	4547	2.347	-2.689	5.23	
										5.25
Test pour 1 période avec Ptolemee, avec un ratio d'environ 90% stérile et 10% minerai	23/08/2000	random.cpp	random	35550	3944	1512	1.609	-2.844	2.49	
				35654	3840	1684	1.551	-2.852	2.48	
				35542	3952	1625	1.592	-2.843	2.47	
				35559	3935	1550	1.549	-2.845	2.47	
				35476	4018	1551	1.613	-2.838	2.49	
	12/09/2000	random.cpp	random	35582	3912	1953	1.593	-2.847	2.34	
				35600	3894	1806	1.591	-2.848	2.39	
				35560	3934	1533	1.576	-2.845	2.40	
				35520	3974	1616	1.606	-2.842	2.44	
				35577	3917	1481	1.559	-2.846	2.32	
										2.43

Tableau A4. 9 : Résultats des tests pour le fichier no 8 pour le graphe inversé

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué
8	686814	65	66	14	60060	39494	oui

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 période avec Ptolemee, avec un ratio d'environ 10% stérile et 90% minerai	24/08/2000	randominv.cpp	random	4022	35472	158	-0.328	14.321	0.66	
				3962	35532	148	-0.317	14.309	0.63	
				4007	35487	150	-0.326	14.352	0.66	
	12/09/2000	randominv.cpp	random	3976	35518	160	-0.318	14.383	0.68	
				3982	35512	164	-0.319	14.389	0.69	
				4003	35491	161	-0.332	14.377	0.72	
				3982	35512	165	-0.319	14.363	0.72	
				3950	35544	170	-0.316	14.362	0.74	
				3854	35640	152	-0.308	14.442	0.68	
				4112	35382	130	-0.329	14.289	0.75	
										0.69
Test pour 1 période avec Ptolemee, avec un ratio d'environ 20% stérile et 80% minerai	24/08/2000	randominv.cpp	random	7883	31611	313	-0.631	12.781	0.69	
				7988	31506	310	-0.639	12.774	0.73	
				7910	31584	336	-0.633	12.758	0.72	
	12/09/2000	randominv.cpp	random	7879	31615	315	-0.630	12.741	0.74	
				7922	31572	314	-0.634	12.705	0.70	
				7775	31719	320	-0.622	12.817	0.73	
				7663	31531	339	-0.637	12.733	0.76	
				7954	31540	330	-0.636	12.719	0.73	
				7938	31556	343	-0.635	12.768	0.72	
				7870	31624	330	-0.630	12.869	1.11	
										0.76
Test pour 1 période avec Ptolemee, avec un ratio d'environ 30% stérile et 70% minerai	24/08/2000	randominv.cpp	random	11822	27672	508	-0.946	11.136	0.71	
				11884	27610	466	-0.951	11.123	0.72	
				11987	27507	491	-0.959	11.176	0.70	
	12/09/2000	randominv.cpp	random	11895	27599	485	-0.952	11.085	1.70	
				11774	27720	496	-0.942	11.211	0.78	
				11763	27731	490	-0.941	11.143	0.80	
				11871	27623	486	-0.950	11.161	1.16	
				11837	27657	454	-0.947	11.157	2.05	
				11985	27509	506	-0.959	11.105	2.01	
				12020	27474	470	-0.962	11.123	0.82	
										1.15
Test pour 1 période avec Ptolemee, avec un ratio d'environ 40% stérile et 60% minerai	24/08/2000	randominv.cpp	random	15720	23774	622	-1.258	9.607	2.03	
				15844	23650	633	-1.267	9.569	0.79	
				15737	23757	652	-1.259	9.622	1.43	
				16016	23478	642	-1.281	9.492	1.27	
	12/09/2000	randominv.cpp	random	15682	23812	647	-1.255	9.625	1.94	
				15783	23711	628	-1.263	9.587	1.88	
				15846	23648	615	-1.268	9.607	2.06	
				15779	23715	624	-1.262	9.526	1.17	
				15893	23601	637	-1.271	9.515	1.99	
				15910	23584	627	-1.273	9.550	1.62	
										1.62

Tableau A4. 9 : Résultats des tests pour le fichier no 8 pour le graphe inversé (suite)

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 période avec Ptolemee, avec un ratio d'environ 50% stérile et 50% minerai	24/08/2000	randominv.cpp	random	19722	19772	810	-1.578	7.981	2.12	
				19754	19740	765	-1.580	7.963	2.10	
				19793	19701	807	-1.583	7.999	2.01	
	12/09/2000	randominv.cpp	random	19792	19702	750	-1.583	7.908	2.04	
				19668	19826	813	-1.573	8.025	2.20	
				19943	19551	858	-1.595	7.894	2.24	
				19673	19821	811	-1.574	7.988	2.40	
				19697	19797	809	-1.576	8.017	2.08	
				19816	19678	824	-1.585	7.945	2.10	
				19885	19609	856	-1.591	7.928	2.15	
										2.14
Test pour 1 période avec Ptolemee, avec un ratio d'environ 60% stérile et 40% minerai	24/08/2000	randominv.cpp	random	23652	15842	1021	-1.892	6.345	2.29	
				23854	15640	1044	-1.908	6.330	2.21	
				23790	15704	997	-1.903	6.377	2.16	
	12/09/2000	randominv.cpp	random	23722	15772	1050	-1.898	6.373	2.42	
				23823	15671	1017	-1.906	6.355	2.27	
				23751	15743	1058	-1.900	6.349	2.34	
				23473	16021	1026	-1.878	6.467	2.39	
				23829	15665	1026	-1.906	6.361	2.32	
				23821	15673	1024	-1.906	6.306	2.30	
				23667	15827	1072	-1.893	6.405	2.33	
										2.30
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	24/08/2000	randominv.cpp	random	27633	11861	1484	-2.211	4.824	2.24	
				27721	11773	1481	-2.218	4.812	2.33	
				27580	11914	1425	-2.206	4.826	2.36	
	13/09/2000	randominv.cpp	random	27702	11792	1789	-2.216	4.755	2.71	
				27547	11947	1397	-2.204	4.806	2.44	
				27619	11875	1632	-2.209	4.792	2.40	
				27664	11830	1528	-2.213	4.756	2.41	
				27740	11754	1458	-2.219	4.728	2.43	
				27635	11859	1678	-2.211	4.771	2.43	
				27595	11899	1849	-2.208	4.796	2.46	
										2.42
Test pour 1 période avec Ptolemee, avec un ratio d'environ 80% stérile et 20% minerai	24/08/2000	randominv.cpp	random	31564	7930	6114	-2.525	3.219	4.87	
				31498	7996	7190	-2.520	3.223	4.77	
				31550	7944	6008	-2.524	3.196	4.90	
	13/09/2000	randominv.cpp	random	31597	7897	6267	-2.528	3.185	5.15	
				31598	7896	6841	-2.528	3.184	4.90	
				31644	7850	6066	-2.531	3.169	4.79	
				31666	7828	4211	-2.533	3.156	4.96	
				31608	7886	7071	-2.529	3.194	6.73	
				31713	7781	7545	-2.537	3.114	5.24	
				31554	7940	5983	-2.524	3.203	5.06	
										5.14

Tableau A4. 9 : Résultats des tests pour le fichier no 8 pour le graphe inversé (suite)

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 période avec Ptolemee, avec un ratio d'environ 81% stérile et 19% minerai	18/09/2000	randominv.cpp	random	32100	7394	12969	-2.568	2.995	9.90	
				31930	7564	6475	-2.554	3.059	5.28	
				32153	7341	10333	-2.572	2.943	9.61	
				32087	7407	8858	-2.567	2.988	5.46	
				32077	7417	9309	-2.566	3.003	7.53	
				31976	7518	8226	-2.558	3.064	7.75	
				31986	7508	10898	-2.559	3.052	7.33	
				31928	7566	9689	-2.554	3.051	7.64	
				31928	7566	9689	-2.554	3.051	7.64	
				31931	7563	8658	-2.554	3.073	7.73	
				31945	7549	9456	-2.556	3.050	7.70	
										7.60
Test pour 1 période avec Ptolemee, avec un ratio d'environ 82% stérile et 18% minerai	18/09/2000	randominv.cpp	random	32357	7137	15428	-2.589	2.868	10.68	
				32413	7081	17221	-2.593	2.878	10.46	
				32485	7009	13321	-2.599	2.844	9.98	
				32449	7045	14881	-2.596	2.862	10.40	
				32393	7101	14954	-2.591	2.869	12.67	
				32525	6969	19730	-2.602	2.814	13.05	
				32403	7091	18375	-2.592	2.892	10.55	
				32307	7187	13736	-2.585	2.898	10.04	
				32403	7091	15279	-2.592	2.881	10.42	
				32381	7113	17092	-2.590	2.856	15.32	
										11.36
Test pour 1 période avec Ptolemee, avec un ratio d'environ 83% stérile et 17% minerai	24/08/2000	randominv.cpp	random	32768	6726	29171	-2.621	2.712	17.87	
				32853	6641	28548	-2.628	2.693	15.13	
				32841	6653	30143	-2.627	2.698	14.65	
	20/09/2000	randominv.cpp	random	32811	6683	30357	-2.625	2.718	21.21	
				32816	6678	24131	-2.625	2.708	15.42	
				32758	6736	26036	-2.621	2.742	17.84	
				32837	6657	24414	-2.627	2.668	17.71	
				32768	6726	30577	-2.621	2.684	17.58	
				32763	6731	21493	-2.621	2.711	13.46	
				32742	6752	28498	-2.619	2.728	21.35	
										17.22
Test pour 1 période avec Ptolemee, avec un ratio d'environ 84% stérile et 16% minerai	20/09/2000	randominv.cpp	random	33253	6241	34061	-2.660	2.524	20.35	
				33159	6335	32279	-2.653	2.558	19.16	
				33197	6297	33798	-2.656	2.549	20.49	
				33234	6260	31062	-2.659	2.516	15.73	
				33322	6172	34691	-2.666	2.510	20.80	
				33136	6358	32009	-2.651	2.580	20.47	
				33088	6406	33438	-2.647	2.596	20.61	
				33245	6249	31701	-2.660	2.548	18.05	
				33272	6222	33677	-2.662	2.522	17.88	
				33296	6198	34564	-2.664	2.518	17.53	
										19.21

Tableau A4. 9 : Résultats des tests pour le fichier no 8 pour le graphe inversé (suite)

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 période avec Ptolemee, avec un ratio d'environ 85% stérile et 15% minerai	24/08/2000	randominv.cpp	random	33444	6050	35180	-2.675	2.426	19.84	
				33552	5942	35474	-2.684	2.407	22.00	
				33570	5924	34026	-2.686	2.399	14.66	
	20/09/2000	randominv.cpp	random	33516	5978	33794	-2.681	2.412	15.63	
				33496	5998	34221	-2.680	2.437	16.68	
				33536	5958	34574	-2.683	2.393	15.70	
				33613	5881	34898	-2.689	2.391	14.00	
				33604	5890	34745	-2.688	2.381	17.84	
				33625	5869	35230	-2.690	2.369	16.97	
				33638	5856	34790	-2.691	2.372	18.05	
										17.14
Test pour 1 période avec Ptolemee, avec un ratio d'environ 87% stérile et 13% minerai	20/09/2000	randominv.cpp	random	34207	5287	36050	-2.737	2.126	17.99	
				34216	5278	37089	-2.737	2.102	11.89	
				34362	5132	37049	-2.749	2.095	15.62	
				34302	5192	36845	-2.744	2.091	14.65	
	21/09/2000	randominv.cpp	random	34351	5143	35874	-2.748	2.096	14.15	
				34380	5114	36397	-2.750	2.074	14.02	
				34441	5053	36303	-2.755	2.055	17.03	
				34334	5160	36970	-2.747	2.085	18.58	
				34374	5120	36669	-2.750	2.073	13.59	
				34430	5064	37093	-2.754	2.037	13.63	
				34298	5196	36841	-2.743	2.077	16.88	
										15.28
Test pour 1 période avec Ptolemee, avec un ratio d'environ 90% stérile et 10% minerai	24/08/2000	randominv.cpp	random	35480	4014	37543	-2.838	1.623	13.18	
				35579	3915	37925	-2.846	1.552	10.61	
				35545	3949	37839	-2.844	1.599	10.39	
	21/09/2000	randominv.cpp	random	35527	3967	38159	-2.842	1.600	11.39	
				35480	4014	37756	-2.838	1.604	11.42	
				35579	3915	38119	-2.846	1.582	10.82	
				35684	3810	37888	-2.855	1.541	10.96	
				35677	3817	37967	-2.854	1.551	10.77	
				35575	3919	37763	-2.846	1.578	14.47	
				35637	3857	38009	-2.851	1.547	11.36	
										11.54
Test pour 1 période avec Ptolemee, avec un ratio d'environ 95% stérile et 5% minerai	21/09/2000	randominv.cpp	random	37548	1946	38994	-3.004	0.794	9.22	
				37569	1925	38885	-3.005	0.787	9.24	
				37525	1969	39022	-3.002	0.787	8.92	
				37448	2046	38957	-2.996	0.824	9.29	
				37458	2036	38844	-2.997	0.812	9.31	
				37430	2064	38915	-2.994	0.838	9.34	
				37543	1951	38857	-3.003	0.799	9.25	
				37487	2007	38800	-2.999	0.810	9.18	
				37562	1932	38827	-3.005	0.781	8.79	
				37506	1988	38920	-3.000	0.820	8.80	
										9.13

Tableau A4. 9 : Résultats des tests pour le fichier no 8 pour le graphe inversé (suite)

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs minerai	nombre de blocs extraits	valeur de la coupe à la source (E08)	valeur de la coupe au puit (E08)	CPU (sec)	Moyenne des CPU
Test pour 1 période avec Ptolemee, avec un ratio d'environ 99% stérile et 1% minerai	24/08/2000 21/09/2000	randominv.cpp randominv.cpp	random random	39114 39085 39128 39083 39088 39101 39129 39113 39128 39101	380 409 366 411 406 393 365 381 366 393	39414 39430 39442 39409 39411 39381 39378 39397 39439 39398	-3.129 -3.127 -3.130 -3.127 -3.127 -3.128 -3.130 -3.129 -3.130 -3.128	0.160 0.167 0.145 0.163 0.161 0.162 0.160 0.159 0.145 0.155	6.22 6.64 6.69 6.61 6.71 6.62 6.65 6.69 6.73 7.09	6.67

Tableau A4. 10 : Résultats des tests pour le fichier no 9 : 768014

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué
9	768014	76	80	14	85120	60004	oui

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	07/08/2000	paramninv.cpp	32	?	?	8795	4.76 4.7 4.69	4.72
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	22/08/2000	random.cpp	random	42112 41924 41807 41939 42065 41848 42071 42134 41998 41963	17892 18080 18197 18065 17939 18156 17933 17870 18006 18041	57257 57468 57560 57271 57561 57435 57317 57345 57581 57408	19.98 18.46 18.33 22.57 19.86 18.41 15.91 18.34 18.32 18.41	18.86

Tableau A4. 11 : Résultats des tests pour le fichier no 10 : 978014

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué
10	978014	97	80	14	108640	79702	oui

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	07/08/2000	paramninv.cpp	32	?	?	11862	6.51 6.54 6.51	6.52
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	22/08/2000	random.cpp	random	55492 55733 55900 55859 55806 55652 55906 55841 55835 55841	24210 23969 23802 23843 23896 24050 23796 23861 23867 23861	76451 76375 76337 76339 76410 76186 76403 76318 76414 76182	30.49 24.68 24.58 32.3 24.92 26.96 21.34 27.03 26.42 18.92	25.76

Tableau A4. 12 : Résultats des tests pour le fichier no 11 : 1357114

	nom du fichier	X	Y	Z	nombre de blocs dans le gisement	nombre de blocs à l'interieur du cône	cône tronqué
11	1357114	135	71	14	134190	99974	oui

Description des test	date	programme	prix (couts)	nombre de blocs stériles	nombre de blocs	nombre de blocs	CPU (sec)	Moyenne des CPU
Test pour 1 periode avec Ptolemee	04/08/2000 07/08/2000	paramninv.cpp id	32 id	? ?	? ?	16018 id	8.52 8.46 8.5 8.34	8.46
Test pour 1 période avec Ptolemee, avec un ratio d'environ 70% stérile et 30% minerai	22/08/2000	random.cpp	random	70048 70136 69914 69813 69937 69804 69877 69799 70034 69791	29926 29838 30060 30161 30037 30170 30097 30175 29940 30183	95773 95818 95766 95696 95777 95593 95653 95803 95725 95820	34.41 42.43 32.22 31.95 27.1 30.95 34.24 30.64 33 30.78	32.77