Analysis of Maximum Flow Algorithms for Ultimate Pit Contour Problems

Michel Gamache

GERAD and École Polytechnique
P.O. Box 6079, station "Centre-ville"
Montréal, Québec, Canada, H3C 3A7
michel.gamache@courriel.polymtl.ca

Geneviève Auger

École Polytechnique
P.O. Box 6079, station "Centre-ville"
Montréal, Québec, Canada, H3C 3A7
genevieve.auger@courriel.polymtl.ca

October, 2000

Les Cahiers du GERAD

G–2000–57

Abstract

The operating schedule problem in open-pit mine can be associated with the problem of maximal closure on a graph. In the literature, the Lagrangian relaxation represents the most interesting approach to solve this problem. The problem resulting from the relaxation of the resource constraints is a sequence of ultimate pit contour (UPC) problems, which ones are solved using a maximal flow algorithm. This paper focuses on the solution of the UCP problem and proposes an analysis of different maximal flow algorithms to solve this problem, taking into account the structure of the graph that are involved. Some accelerating strategies are described and analyzed.

Résumé

Le problème de fabrication d'horaire de production dans les mines à ciel ouvert peut être associé à un problème de fermeture de poids maximal dans un graphe. Dans la littérature, la relaxation Lagrangienne représente la méthode la plus intéressante pour résoudre ce problème. Le problème résultant de la relaxation des contraintes de ressource consiste à identifier une séquence de contours ultimes, où chaque contour peut être trouvé par la résolution d'un algorithme de flot maximum. Cet article se concentre plus particulièrement sur la résolution du problème des contours ultimes et propose une analyse des algorithmes de flot maximum et des stratégies utilisées pour accélérer la résolution du problème, considérant la structure particulière des graphes impliqués.

1 Introduction

An important aspect of the open-pit mine planning is the elaboration of the best production schedules, i.e. the schedules that will bring the biggest revenues. This problem has been well covered in the literature. Among the methods that have been proposed, those based on the Lagrangian relaxation are the most interesting ones because the relaxation of some specific constraints converts the initial problem into a sequence of UPC problems for which efficient solution methods are well known.

The Lagrangian relaxation is an iterative approach. At each iteration, a series of UPC problems are solved based on the current values of some parameters that influence each pit contour. The solution time of this approach depends on both the way the values of the parameters are adjusted and the time spent solving each UPC problem. This paper deals only with this latter element. In order to better understand the structure of the problem and identify the key elements of a good solution approach, an analysis of different maximum flow algorithms and strategies is proposed. In the second section, the approach that has led to the Lagrangian relaxation is briefly presented. The third and fourth sections describe augmenting path and preflow-push algorithms, respectively. In each case, accelerating strategies are discussed and analyzed in relation with the graph involved in the UPC problem. Finally, future research directions conclude this paper.

2 Problem Description

In its simplest form, the problem of designing the best contour of an open pit mine, called the ultimate pit contour, consists in identifying the set of blocks to extract in order to maximize the total revenue for one period. This problem can be easily formulated as a linear integer program. Let n be the number of blocks in the geological model, c_i the revenue associated with the extraction of the i^{th} block ($i = 1, \ldots, n$), and Γ_i the set of blocks that have to be extracted before the extraction of i. Finally, let x_i be the decision variable associated with each block, where:

$$x_i = \begin{cases} 1 & \text{if block } i \text{ is extracted,} \\ 0 & \text{otherwise.} \end{cases}$$

The mathematical formulation of the UPC problem can be written as follow:

$$\begin{aligned} &\max \sum_{i=1}^{n} c_i x_i \\ &s.t. \\ &\qquad x_i - x_j \leq 0, \qquad j \in \Gamma_i,\ i = 1, \ldots, n \\ &\qquad\quad x_i \in \{0,1\} \quad i = 1, \ldots, n. \end{aligned} \tag{1}$$

In this formulation, the objective function maximizes the revenue generated by the blocks that have been selected and the constraints (called the *precedence constraints*) ensure

that all the blocks included in Γ_i have been selected when block i is selected. Let call this problem P1.

Because of the large number of binary variables, solving problem P1 by using any implicit enumeration or branch-and-bound techniques would be ineffective. The combinatory will introduce very long solution time. However, other approaches are possible; in fact, this problem is similar to finding the maximum closure of a directed graph $G(V, A)$. A *closure* is a set of nodes such that if a node belongs to this set, then all its successors also belong to the set. If a real number is associated with each node, then the maximum closure is the set of nodes with the biggest value. For the UPC problem, the graph $G(V, A)$ consists of $V = \{1, 2, \ldots, i, \ldots, n\}$ a set of nodes (vertexes, blocks) and $A = \{(i, j) | i \in V,\ j \in \Gamma_i\}$ a set of arcs representing the link between blocks given by the precedence constraints.

Picard (1976) has shown that the problem of finding the maximum closure of an acyclic graph $G(V, A)$ can be modeled as a maximum flow problem on a particular graph, called in this paper Picard's graph and noted $\mathcal{G}(\mathcal{V}, \mathcal{A})$. Given $V^+ = \{i \in V | c_i > 0\}$ and $V^- = \{i \in V | c_i \leq 0\}$, the new graph $\mathcal{G}$ is defined by $\mathcal{V} = V \cup \{s, t\}$ and $\mathcal{A} = A \cup A^+ \cup A^-$, where $A^+ = \{(s, i) | i \in V^+\}$ and $A^- = \{(i, t) | i \in V^-\}$. The capacity on arcs is equal to c_i for all arc $(s, i) \in A^+$, to $|c_i|$ for all arc $(i, t) \in A^-$ and finally to ∞ for all arc $(i, j) \in A$. Fast and efficient algorithms can be used to solve the maximum flow problem. Some algorithms will be discussed in the next two sections.

Now, what happens when one wants to find the best sequences of production; i.e. when the revenue associated with the extraction of a block is based on the period of extraction? Let call this new problem P2. Based on the mathematical formulation of P1, the introduction of multiple periods in P2 involves the addition of a second index for the variables and constants. Let $p = 1, \ldots, P$ be that index and P be the number of periods of production. Moreover, the mathematical model necessitates the introduction of a new set of constraints that ensures that each block is extracted during only one period. A description of the mathematical model can be found either in Dagdelen & Johnson (1986) or Tachefine (1997). The main drawback with the precedent mathematical formulation is that the structure of the maximum flow problem disappears. But, Dagdelen & Johnson have shown that it is possible to modify the formulation and retrieve the structure of the maximum flow problem. In the new formulation, it is possible to construct a multi-layer graph (each layer representing a period) on which a maximum flow problem can be solved. However, this large-scale graph represents a major difficulty in practice for the solution of this problem. To overcome this difficulty, Dagdelen & Johnson have proposed an interesting decomposition of the multi-layer graph that permits to sequentially solve each layer without lost of generality. Their idea consists in solving a maximum flow problem for each layer of the graph, i.e. solving a problem P1 for each period, starting with the layer representing the first period and transferring information (flows and capacities) to the next period (next layer in the graph). Tachefine have proved that the transfer of information proposed by Dagdelen & Johnson was not always optimal and he suggested a variation of this transfert that is optimal.

The solution approach proposed by Johnson & Dagdelen makes the use of efficient maximum flow algorithms still possible even if more than one period are considered. The

solution of this problem will produce a set of contours, one for each period considered in the scheduling time horizon. But in real life operations, the problem of finding the best sequences of operation is subject to operations constraints like limitations on equipment production capacity, blending constraints, ore/waste ratio, etc. Let call the new set of constraints the *resources constraints* and this new problem P3. Let also $AX \leq b$ be the general form of the resource constraints. This new set of constraints destroys the structure of the problem P2 and makes impossible the use of maximum flow algorithms. In order to retrieve the structure of P2, a Lagrangean relaxation of the resource constraints is applied; i.e. the set of resource constraints is removed and the term $\lambda(AX - b)$ is subtracted from the original objective function. Since λb is a constant it can be ignored in the new objective function. Each resource constraint k is multiplied by a parameter λ_k called a lagrangian multiplier. Then for solving problem P3, an iterative procedure is used where the lagrangian multipliers are adjusted at each iteration. When a resource constraint for a specific period is violated, its multiplier increases or reduces the revenue associated with each block of the period, which will affect the contours of the mine at the next iteration. For the adjustment of the multipliers, people use subgradient, cutting plane and bundle methods. In order to speed up the solution, some authors propose heuristic approach for finding feasible solution. Since these approaches provide excellent upper bounds but poor feasible solutions, greedy heuristics are proposed for finding good feasible solutions.

At each iteration of the adjustment procedure, P maximal closure problems (P1) are solved sequentially. The efficiency of the algorithm used to solve this problem is quite important. The next two sections deal with solution approaches using maximal flow algorithms.

3 Augmented Path Algorithms

The algorithms that have been developed so far to solve the maximum flow problem can be grouped into two classes of strategies : *augmented path* and *preflow-push* algorithms. The augmented path algorithms will be described in this section while the next section will present the preflow-push algorithms. In this paper, algorithms are briefly presented, but more details on them and their complexity analysis can be found, for example, in Ahuja et al. (1993).

In both cases, the algorihms will always be solved on a graph, called *residual graph*, associated with $\mathcal{G}$. A residual graph, noted here $\mathcal{G}^r(\mathbf{x})$, is associated with graph $\mathcal{G}$ for the current flow array $\mathbf{x}$. For each arc (i, j) in the original graph $\mathcal{G}$ having a flow x_{ij}, the residual graph $\mathcal{G}^r(\mathbf{x})$ contains at most two arcs: one arc (i, j) having a residual capacity $r_{ij} = c_{ij} - x_{ij}$ and one arc (j, i) having a residual capacity $r_{ji} = x_{ji}$. Moreover, the residual graph $\mathcal{G}(\mathbf{x})$ only contains arcs such that $r_{ij} > 0$.

As mentioned by Yegulap & Arias (1992), based on the definition of $\mathcal{G}^r$ air blocks or waste blocks with zero value won't be part of $\mathcal{G}^r$ since $r_{ij} = 0$ for these blocks. Hence, the construction of $\mathcal{G}^r$ will implicitly reduce the graph.

Each iteration of the generic augmented path algorithm consists of two main steps. In the first one, the algorithm tries to identify a path $\mathcal{P}$ from node s to node t in $\mathcal{G}^r(\mathbf{x})$,

where $\mathbf{x} = \mathbf{0}$ at the beginning. When such a path is found, the second step consists in augmenting the flow on this path by a value δ that corresponds to the residual capacity of the path, i.e. $\delta = \min\{r_{ij} | (i,j) \in \mathcal{P}\}$. The new flow array $\mathbf{x}$ will modify the residual graph. The procedure is repeated until no directed path exists between s and t; then, the optimal solution is found.

This algorithm runs in $O(mnU)$ time, where U represents an upper bound on arc capacity in the graph. This complexity is composed of two terms : the search for a path and the number of times these searches will occur (i.e. the number of flow augmentations). The search for a feasible path from s to t is bounded by $O(m)$ the number of arcs in the graph. The number of possible augmenting paths has a bound of $O(nU)$, which is the upper bound on the capacity of a cut $[s, V \backslash s]$.

The complexity of algorithms is based on worst case scenarios. What is the real complexity when it is applied on $\mathcal{G}^r(\mathbf{x})$? At first sight, this complexity may looks disastrous for the UPC problem since the capacity on arcs $(i,j) \in A$ is a very large number (theoretically this capacity is set to infinity). However, these arcs won't be part of the minimal cut; otherwise, we would have an unbounded problem. So for the following explanations, we will consider U as the upper bound on the absolute value of a block in the mine, i.e. the maximal capacity among outgoing arcs of node s (arcs $(i,j) \in A^+$) or incoming arcs of node t (arcs $(i,j) \in A^-$). In practice we know that the number of augmenting path will be bounded by $O(\beta)$ where $\beta = \min\{\beta_1, \beta_2\}$, $\beta_1 = \sum_{i \in V^+} c_i$ and $\beta_2 = \sum_{i \in V^-} |c_i|$. Since $\beta \ll nU$ it represents a better upper bound.

Improvements have been proposed to speed up the solution time of augmented path algorithms. One of these improvements consists in restricting the choice of augmenting paths to the shortest ones. This strategy tries to reduce the time spent on the search for a path. It relies on a label, noted $d(i)$, that indicates the lower bound on the shortest distance in $\mathcal{G}^r(\mathbf{x})$, in terms of the number of arcs, from node i to node t. An arc (i,j) is defined *admissible* if it satisfies both conditions $r_{ij} > 0$ and $d(i) = d(j) + 1$. Similarly, an *admissible path* will only contain admissible arcs. In such condition, the label $d(s)$ represents a lower bound on the length of a path from the source to the sink in the residual network. Hence, any admissible path from s will be a path with the shortest number of arcs. Node labeling is done by applying a reverse breadth-first search on $\mathcal{G}^r$ from node t; i.e. $d(i) = \min\{d(j) + 1 | (i,j) \in \Gamma_i\}$.

At each iteration, the algorithm finds the shortest path in the residual graph; performs an augmentation of flow on this path; updates the residual graph; and, relabels some nodes. It has been proved that the optimal solution is found when $d(s) \geq n$. Since the algorithm stops when $d(s) \geq n$, then the longest path encountered in the solution process will contains at most n arcs, which decreases the complexity for the search of a path from $O(m)$ to $O(n)$.

In practice, it has been proved that it is possible to stop the algorithm before $d(s) = n$ by using a n-dimensional array, called *list*, that will compute the number of nodes having a label equal to k, $(k = 0, ..., n-1)$. If $list(k) = 0$ for a specific $k < d(s)$, then the algorithm stops because it implies that there is no admissible path from s to t in the

residual graph. Moreover, this gap represents the minimum cut in the graph; i.e., all nodes i having $d(i) > k$ will be part of the set of nodes containing s, while nodes j which $d(j) < k$ will be part of the set of nodes containing t. All the nodes in the set containing s will represent the blocks in the mine that will be extracted.

In order to show how this new stopping criterion improves in practice the solution of the UPC problem, we will first formulate three propositions.

Proposition 1: A directed path $\mathcal{P}$ in $\mathcal{G}^r$ between s and t must contain at least an arc (i, j) where $i \in V^+$ and $j \in V^-$.

Proof: Since $\forall i \in V^+ \; \exists (s, i)$ and $\nexists (i, t)$, $\forall j \in V^- \; \exists (j, t)$ and $\nexists (s, j)$, then a path between s and t must at least contain an arc (i, j) where $i \in V^+$ and $j \in V^-$. ■

Proposition 2: $\forall i \in V^-$ the distance label is initialized at $d(i) = 1$.

Proof: From the definition of A^- we know that all nodes $i \in V^-$ are linked with the sink node t by an arc (i, t) which implies $d(i) = 1$. ■

Proposition 3: If there is a path $\mathcal{P}$ from s to t, then the distance label of node s is initialized at $d(s) = 3$.

Proof: Since node s is linked only to nodes $i \in V^+$, any path $\mathcal{P}$ contains at least one arc (s, i). Since only nodes $j \in V^-$ are linked to node t, then any path $\mathcal{P}$ contains at least one arc (j, t). Finally, since there is at least one arc (i, j) in the graph, then it is possible to construct a path $s \to i \to j \to t$ in the graph, which implies that $d(s) = 3$. ■

Since $d(s) = 3$ at the beginning of the solution approach and $3 \ll n$, it is easy to figure out that it would have taken a lot of relabeling operations before reaching the stopping criterion $d(s) = n$. Thereby, the use of the array *list* speeds up the identification of the minimum cut, which is the only aspect of the maximum flow problem solution that is important for the UPC problem.

Moreover, the use of a distance label on nodes permits implicitly to reduce the graph. To explain this, let first formulate two new propositions.

Proposition 4: Let the set $\mathcal{C} \subseteq V^+$ be a closure in $\mathcal{G}$. Then $\forall i \in \mathcal{C}$ there is no path from i to t in $\mathcal{G}$.

Proof: If $i \in \mathcal{C}$ then any $j \in \Gamma_i$ is also $\in \mathcal{C}$, i.e. $\in V^+$. From the definition of $\mathcal{G}$, all nodes $i \in \mathcal{C}$ will either be linked to s or to another node in $\mathcal{C}$. Then from proposition 1 we can conclude. ■

The set $\mathcal{C}$ represents a closure with a positive weight, then all the blocks included in $\mathcal{C}$ will be extracted, which means that all the nodes $i \in \mathcal{C}$ will be part of the same set as node s in the minimal cut.

Proposition 5: $\forall i \in \mathcal{C}$, $d(i)$ can be set equal to n.

Proof: It is known from proposition 4 that in the labeling process, these nodes won't receive a label since it is impossible to reach node t by a directed path from these nodes. Since they are part of a positive weight closure, each node $i \in \mathcal{C}$ should be extracted, i.e. should have a label $d(i) \geq n$ at the end of the solution process. By setting the label of these nodes at n, they won't be considered in the solution process, except at the end. ■

From the latest proposition, we can see that any graph reduction trying to remove positive-weight blocks on the upper levels (sets of blocks that form closures) will not result in savings on solution time in an algorithm using distance labels since these nodes will be automatically set to n in the preprocessing.

4 Preflow-Push Algorithms

For the last ten years, many preflow-push algorithms have been proposed for solving the maximum flow problem. We present here the algorithm developed by Goldberg & Tarjan (1988) and different strategies based on this algorithm that have been used to reduce the solution time.

In the preflow-push algorithms, instead of sending flow along paths from source to sink, flows are sent on individual arcs which causes the violation of the flow conservation constraint at each node. At each iteration, the solution is infeasible. The optimal solution is obtained when the flow conservation constraint is respected at each node. A preprocessing procedure is first used and consists in pushing flows on arcs outgoing of the source node. The amount of flow on each arc is equivalent to the capacity of the corresponding arc. This procedure results in an excess of flows, noted $e(i)$, at each node $i \in V^+$ because the flow entering nodes i equals c_i and the one leaving these nodes is zero (the flow conservation constraint at these nodes is not respected). During the algorithm, every node, except the source node, will either be in *excess* ($e(i) > 0$) or *balanced* ($e(i) = 0$). Nodes with excess are called *active nodes* and by convention s and t are never considered active. Let V^a be the set of active nodes. In the following procedures, the algorithm tries to push flows from active nodes through the graph up to the sink node. Distance labels on nodes are also used in the generic preflow-push algorithm. After the preprocessing phase, the source node is immediately labeled $d(s) = n$. A push of δ units of flow on an admissible arc (i, j) will be called a *saturating push* when $\delta = r_{ij}$ and *non-saturating* otherwise. When a non-saturating push is done on arc (i, j), node i leaves the set V^a while j enters this set if $j \notin V^a$. If an active node has no admissible arc, then its distance label is increased (relabeled : $d(i) = \min\{d(j)+1|(i, j) \in \Gamma_i\}$). This procedure will permit to push the excess of flow back. When no more excess can be pushed to the sink node, then the algorithm will push the excess back to the source node in order to retrieve a feasible solution. This return process is not essential in many problems, especially when people are only interested in the minimal cut.

The number of relabeling steps that actually change node labels runs in at most $O((n-1)^2)$, the number of saturating pushing steps is at most $O(nm)$ and the number of non-saturating pushing steps is at most $O(n^2m)$. Thereby, the generic preflow-push algorithm runs in $O((n-1)^2 + nm + n^2m+) = O(n^2m)$ time, which is the bound given on the maximum number of non-saturating pushes that can be performed. We will see later that this bound may be not really representative of the solution time in practice when this algorithm is applied on $\mathcal{G}$.

Many strategies have been proposed in order to stop the algorithm sooner. We only discuss the approach based on distance label, which is similar to the one presented for the

augmented path algorithms. The approach consists in using the n-dimension array *list* that counts the number of labels having a distance equal to k. Let $L < n$ be the highest distance label for which $list(L) > 0$. Whenever there is $k \leq L$ which $list(k) = 0$ then any node i with $d(i) > k$ is disconnected from the set of nodes j with $d(j) < k$; i.e. there is no path between these nodes i and t. So as for the augmented path algorithms, we can increase the label of nodes i to n. This strategy is called *gap relabeling* by Hochbaum & Chen (1998).

Once again the use of the distance label permits implicitly to reduce the graph. Let define the set $\mathcal{W} = \{j \in V^- | \forall (i,j) \in \mathcal{G},\ i \in V^-\}$ and formulate two new propositions.

Proposition 6: Any node j such $j \in \mathcal{W}$ will always remain inactive during all the solution process.

Proof: Since these nodes do not have positive weight predecessors, no push will be done to these nodes. Thus, their excess will always remain equal to 0, which means that they will be out of V^a, the set of active nodes. ■

Proposition 7: In the minimal cut, all node $j \in \mathcal{W}$ will be in the same set as t.

Proof: Since these nodes will never enter in set V^a, then they will never be relabeled. Thus, their label will remain equal to 1. ■

Similarly to the positive weight closure $\mathcal{C}$ presented in the previous section, the set $\mathcal{W}$ represents the set of blocks having negative weights that are located on the lowest levels of the geological model. Trying to remove these blocks in preprocessing, while using a preflow-push algorithm, will not result in time-savings since these blocks will be ignored during the solution process.

The way active nodes are selected in set V^a may also influence the speed of the algorithm. In the literature, three strategies are often mentioned and compared: First In First Out (FIFO), Last In First Out (LIFO), and the Highest-Label (HL).

The FIFO consists in treating active nodes one after the other in the order they have been created. The first active node on the front of the list V^a is selected and removed of V^a. If that node has an admissible arc (i,j) then δ units of flow are pushed on this arc. If it is a non-saturating push, node i is removed from V^a and node j is added at the rear of V^a. If the push is saturating and i is still active, then i is still considered and another admissible arc must be found; and j is added at the rear of V^a. If no admissible arc is found, node i is relabeled and add to the rear of the list.

Hochbaum & Chen (1998) mention the use of a LIFO strategy. Let node i be the current node. If a non-saturating push is done on arc (i,j), node i is removed from V^a and node j is added at the front of V^a and becomes the new current node. If a saturating push is done on the arc (i,j), then the node j becomes active and thereby the new current node, and this even if i remains active.

Hochbaum & Chen (1998) compared the FIFO and LIFO strategies and obtained the best solution time with a preflow-push algorithm using a LIFO strategy combined with a gap-relabeling strategy. With the LIFO strategy, the push-relabel actions are repeatedly applied to the node to the top of the list V^a, no action is performed on the nodes in the rest of the list until they rise to the top. According to Hochbaum & Chen: "... when the

LIFO strategy is used, the values of the distance labels may extend over a wider range and may be more scattered than when the FIFO strategy is used. This make the occurrence of gaps more likely with the LIFO strategy, thus gap-relabeling pushes up labels faster and speeds up the algorithm."

The LIFO strategy can be seen as a depth-first search in the network (if we consider the sink node as the deepest node in the network) while the FIFO strategy will be much closer to a breadth-first exploration. The LIFO strategy will tend to push flow from the active nodes in the list V^a that are always the closest to the sink node. Considering that, it is difficult to figure out how the distance labels may extend on a wider range as mentioned by Hochbaum & Chen. Instead, we believe that since the LIFO strategy works on active nodes having among the smallest distance labels that the gap may occur at a level k that is closer to 0, which will be quite efficient since all node having $d(i) > k$ will be relabeled at $d(i) = n$. By opposition, since FIFO strategy works with active nodes having different distance labels, it will not necessarily produce a gap.

Both strategies, FIFO and LIFO, do not reduce explicitly the number of non-saturating pushes, which is, theoretically, the bottleneck operation for the preflow-push algorithms. The third strategy, HL, works on this aspect of the problem. This strategy consists in ordering active nodes from those having the highest distance label to those having the smallest one. By pushing flow first from the active nodes that are the farthest from the sink node, then a larger quantity of flow is available for each push on arcs. The main idea behind this strategy is to reduce the number of non-saturating pushes on arcs. Ahuja et al. (1993) mention that the HL preflow-push algorithm is currently the most efficient method ($O(n^3)$) in practice for solving the maximum flow problem because it performs a least number of non-saturating pushes.

Tachefine (1997) compared the FIFO and the HL strategies. The best solution times were obtained with the HL strategy. Tachefine noted that the difference between the solution time increases with the size of the graph. Moreover, Tachefine notices for both strategies that the solution time increases linearly with the size of the graph contrary to what it is expected from the complexity (i.e. $O(n^3)$). Since Tachefine increases his graphs by increasing the number of levels, it is difficult to know if the increase of the solution time is related to n, the number of blocks, or to ℓ, the number of levels. In order to verify this observation, we have done different tests. A first series of tests were done on five different models, each one containing 100,000 blocks but having a different number of levels. Results indicated that the solution time increased linearly with the number of levels. A second series of tests have been done with block models having the same number of levels but having different number of blocks. We obtained similar results as those observed by Tachefine.

Another accelerating strategy has been proposed by Hochbaum & Chen (1998). Their strategy consists in solving the maximum flow algorithm (using the preflow-push approach) on a reverse graph. Let $\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{A}})$ be the reverse graph where $\overline{\mathcal{V}} = \mathcal{V}$ and $\overline{\mathcal{A}} = \{(i,j)|(j,i) \in \mathcal{A}\}$. With this strategy, instead of pushing flow from positive weight blocks to non-positive weight blocks, the opposite is done. As mentioned by these authors, when the capacity of the cut $[s, V\backslash s]$ is larger than the cut $[V\backslash t, t]$, then most of the flow

contributes to keeping nodes active, and the algorithm does not terminate until the label of these nodes exceeds n. By using a reverse graph in such a case (which should always occur in UPC problems), most of the pushes from the sink to the source will be non-saturating pushes, which will result in reducing more rapidly the set of active nodes V^a and thus reaching more rapidly the stopping criterion. We did some tests using this strategy and the results confirmed the results obtained by Hochbaum & Chen.

These results seem in contradiction with the complexity analysis, which indicates that the number of non-saturating push operations should be reduced. One reason that may explain the efficiency of this strategy is that it may avoid pushing flow on precedence arcs which capacity is set to infinity; i.e. all the pushes on these arcs, which represent something like 90% of the arcs in the graph, will be non-saturated. Moreover, the analysis of complexity of the non-saturating push operations (see Ahuja et al., 1993) depends of the number of active nodes and the maximal value of the distance label in the graph, and a reduction of these two elements may indirectly influence the complexity of the non-saturating push operation. Finally, the good results of the preflow-push algorithm on graph $\mathcal{G}^r$ tend to prove that the UPC problem does not represent the worst case scenario.

5 Conclusions

From the analysis and the tests that we have done and the results obtained in the literature, preflow-push algorithms using distance label implementation are among the most efficient method to solve the UPC problems. As indicated, graph reductions in preprocessing phases may sometime be inefficient since these reductions are often implicitly taken into account in the preflow-push algorithms. Future tests will have to be done in order to compare the efficiency of the LIFO and the HL strategies for the selection of active nodes. The reverse graph strategy proposed by Hochbaum & Chen is very interesting and quite efficient for the UPC problem.

It seems clear from our results and those observed in the literature that the UPC problem does not represent the worst case for the preflow-flow algorithms. Since the type of graphs involved on UPC problems is always the same and has a well established structure, we believe that a better analysis of the complexity, adapted for graph $\mathcal{G}$, should be done. For example, the search for a path, the number of relabeling operations and the distance labels depend on n the number of blocks; we believe, without proving it however, that ℓ, the number of levels (or benches), is a better indicator of the solution time. Such analysis is important because it will permit to have a better understanding of the problem and algorithms involved and will lead to the development of adapted algorithms for the UPC problem. Our future works are oriented in that direction.

References

[1] Ahuja, R.K., Magnanti, T.L. and Orlin, J.B., 1993, "Network Flows – Theory, algorithms and applications," *Prentice Hall*, 166–242.

[2] Dagdelen, K. and Johnson, T.B., 1984, "Optimum Open Pit Mine Production Scheduling by Lagrangian Parameterization," *Proceedings of the 19th APCOM Symposium*, 127–142.

[3] Goldberg, A.V. and Tarjan, R.E., 1988, "A New Approach to the Maximum Flow Problem," *J. Assoc. Comput. Mach.*, 35, 921–940.

[4] Hochbaum, D.S. and Chen, A., 1998, "Performance Analysis and Best Implementations of Old and New Algorithms for the Open-Pit Mining Problem," available at: `http://www.ieor.berkeley.edu/ hochbaum/html/publications.html`, pp. 40.

[5] Picard, J.-C., 1976, "Maximal Closure of a Graph and Applications to Combinatorial Problems," *Management Science*, 22(11), 1268–1273.

[6] Tachefine, O.B., 1997, "Méthode d'optimisation pour la planification de la production dans une mine à ciel ouvert," *Ph.D. Thesis, École Polytechnique de Montréal*, January 1997. (*in French*)

[7] Yegulalp, T.M. and Arias, J.A., 1992, "A Fast Algorithm to Solve the Ultimate Pit Limit Problem," *Proceedings of the 23rd APCOM*, 391–397.