# An Open-Source Program for Efficiently Computing Ultimate Pit Limits: MineFlow

Matthew Deutsch[1], Kadri Dağdelen[2], and Thys Johnson[3]

[1] Graduate Student Mining Engineering Department, Colorado School of Mines, Golden Colorado, United States. Email Address: matthewvdeutsch@gmail.com (*corresponding author*) ORCID: 0000-0002-9895-3509
[2] Professor Mining Engineering Department, Colorado School of Mines, Golden Colorado, United States. Email Address: kdagdele@mines.edu
[3] Research Professor, Mining Engineering Department, Golden Colorado, United States. Email Address: tbj1934@comcast.net

**Abstract**

Solving the ultimate pit problem remains an important and relevant task in open pit mine planning as a subroutine in sophisticated decision processes, parametric analysis, quantifying the impact of geologic uncertainty, approaching more complicated problems such as production scheduling, or as a subproblem in decomposition. Ultimate pit models often contain dozens of millions of blocks and hundreds of millions of precedence constraints and they may need to be solved hundreds or even thousands of times. These requirements can quickly overwhelm existing ultimate pit solvers. This paper introduces MineFlow: an open source c++ library providing efficient and flexible precedence schemes and a stripped down pseudoflow-based solver. In a computational comparison with five modern commercial packages we show that MineFlow consistently computes identical results in a fraction of the time; with MineFlow a block model containing 16 million blocks is processed in nine seconds when the fastest commercial alternative takes two and a half minutes. These improvements are realized by using implicit precedence schemes and slightly modifying the conventional pseudoflow algorithm. Additionally, we present a compact notation for the pseudoflow algorithm to assist educators in presenting the latest in ultimate pit optimization.

## Article Highlights

- A capable and computationally efficient open-source solver for the ultimate pit problem based on pseudoflow is introduced
- Novel modifications to the pseudoflow method to improve computational performance are described
- A compact notation for the pseudoflow algorithm is introduced to promote teaching the latest in ultimate pit optimization

**Keywords:** Open pit mining; Ultimate pit problem; Long-term planning; Pseudoflow, Optimization

**Declarations**: Author Matthew Deutsch is employed by one of the anonymous commercial packages used in the computation comparison.
**Funding:** This work was funded by Newmont Corporation
**Conflicts of interest**: The authors have no relevant financial or non-financial conflicts of interests to disclose.
**Availability of code:** All code used to generate precedence graphs and compute ultimate pits is available from the corresponding author and at the following link: https://github.com/MineFlowCSM/MineFlow
**Acknowledgements:** We thank Dr Marcelo Godoy for sponsoring this project. We also thank Dr Conor Meagher and Dr Ryan Goodfellow, also from Newmont, for useful discussions. Additionally, we thank Lee Jornet Zamalloa Llerena, Graduate Student from the Mining Department at Colorado School of Mines, for assisting us in data conversion and running the commercial packages with the different datasets.

## 1 Introduction

Open pit mines are complicated to plan and execute in a manner that uses the earth's resources in a sustainable manner. Mining engineers, and other technologists in the mining industry, are required to call on a diverse toolkit from a wide variety of disciplines to make the best decisions on a day-to-day basis. Operations research has historically been an important discipline in this toolkit as it uses concrete, justifiable, techniques based on mathematical modeling, statistical analysis, and optimization. One of the most foundational problems for which operations research has been used in mining is the ultimate pit problem.

The ultimate pit problem is, at first look, concerned with determining the final pit contour such that the subset of material providing the maximum profit is extracted, and all unnecessary waste is left in place – Figure 1. This problem has borne the brunt of legitimate criticism in recent years. The ultimate pit ignores many of the complexities and realities of mining including the time value of money, mine and mill capacities, stockpiling, selectivity, and more. It has a very narrow focus and the ultimate pit as identified at the beginning of a project will almost certainly not be the final extraction limits in part because certain realities cannot be captured in the initial model and because inputs will continue to change over time.

Despite this, the narrow focus of the ultimate pit problem makes it useful as a subroutine in more complicated pursuits such as scheduling (Buelga Díaz et al 2021), sensitivity analysis, infrastructure placement (Deutsch et al 2015), and the broader circular nature of long-range open pit mine design (Darling 2011). For example, the ultimate pit depends in part on operating costs, those operating costs depend on what equipment is used, and the best equipment to use depends on many factors including the total volume to be extracted, the production rate, and lifetime of the mine which is dependent on its ultimate size. Open pit planners then use iterative procedures throughout the design process, within which the ultimate pit problem remains relevant.

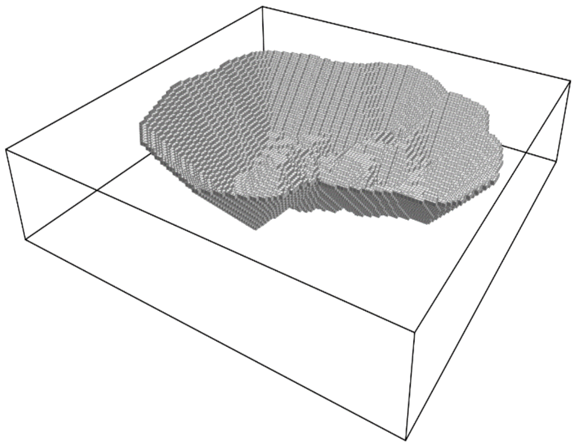

**Fig. 1** An ultimate pit calculated with MineFlow

The straightforward nature of the ultimate pit problem allows for sophisticated decision processes to be built on top of it and carries with it a mathematical structure that enables efficient algorithms. Historically implementations of these algorithms have been locked within expensive commercial packages or restricted behind non-permissive software licenses. This software landscape serves to limit productivity and research output. This holds the mining industry back by limiting innovation and making creative applications of the ultimate pit problem difficult.

This paper introduces MineFlow, an open-source software package for defining precedence graphs and solving the ultimate pit problem. MineFlow is provided by the Colorado School of Mines and Newmont Corporation under the permissive MIT license which ensures that it is free to use in both academic and commercial settings. The code is available at https://github.com/MineFlowCSM/MineFlow. It is open source to invite improvements and contributions from researchers and users. The underlying solver is a stripped-down reimplementation of the Pseudoflow algorithm (Hochbaum, 2008). However, our implementation has been designed to focus solely on the ultimate pit problem and places a heavy emphasis on speed and low memory use. MineFlow is suitable for block models with hundreds of millions of blocks and billions of implicit precedence constraints.

In the next section we will give a thorough overview of the ultimate pit problem and revisit the required theory and necessary background. In Section 3 we describe the pseudoflow algorithm along with our modifications in detail and go through a small example which introduces our shorthand notation. We believe that this compact notation is approachable and will help educators to clearly communicate the principles behind the pseudoflow algorithm. Section 4 describes our open-source implementation and compares it with available commercial implementations.

## 2 Background

### 2.1 Block Precedence

Conventional open pit mine design is simplified by dividing the operational area of interest into a regular 3D block model. Each block within the model contains numeric and categorical attributes which are estimated or simulated through geostatistical processes. These local attributes and other global parameters are then used to calculate an economic block value for each block which roughly indicates the profit, often negative, if that block were mined. This is not a simple calculation and there are many considerations, for our purposes we will consider the appropriate discretization and valuation of the deposit as known inputs.

The remaining parameter of interest is the precedence constraints which dictate the shape of the ultimate pit. Unlike underground mines, which use a variety of techniques to support the material above the deposit, open pit mines proceed downwards removing successive layers of material in a roughly cone shape. The slope of the cone is based on the strength of the material being mined, its geotechnical properties, and other structural geologic factors. This slope will generally vary by location and direction throughout the mining area.

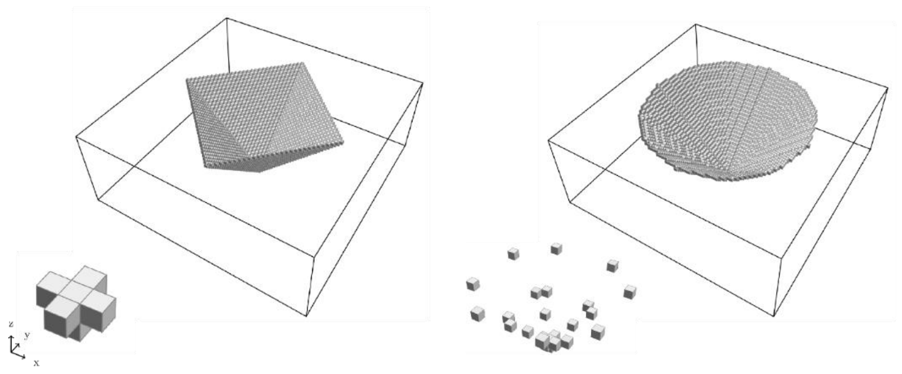

**Fig. 2** Left: The 'One-five' precedence pattern extended 30 blocks vertically. Right: the 29 block min search precedence pattern extended 30 blocks vertically

In their seminal 1965 paper, Lerchs and Grossmann proposed the 'One-five' precedence pattern where each block is connected to the five blocks immediately above it in a cross pattern. This scheme is locally precise but globally inaccurate as seen on the left in Figure 2. Better precedence schemes are described in Caccetta and Giannini (1988) where they introduce a technique for generating efficient 'minimum search patterns' wherein the pit slopes can vary by direction and the pattern consists of a bare minimum of precedence constraints. The 45-degree minimum search pattern for 9 benches is shown on the right in Figure 2, and when this pattern is extended vertically it closely resembles a cone.

Further precedence schemes are described in Chen (1976), Khalokakaie and Dowd (2000), and Gilani and Sattarvand (2015). This is a common topic in the literature but again we sidestep much of this discussion and take the precedence constraints as given inputs.

### 2.2 Ultimate Pit Problem

The ultimate pit problem is naturally formulated as a linear programming problem. We use a single variable $X_b \in B$ which represents the proportion of each block that is mined where $B$ is the set of all blocks. Our single parameter, $v_b \in B$, is the economic block value described above. Our objective is to maximize the product of $v_b$ and $X_b$; our final mined value. The first set of constraints are the precedence constraints. These define for each block, $b$, what other blocks, $\hat{b} \in \widehat{B_b}$, must be mined before $b$. Note that the set of target blocks $\widehat{B_b}$ varies for each base block. The final set of constraints ensure the proportion of each block mined is between zero and one. It is not necessary to restrict the values of $X_b$ to be integer, they will only assume integer values due to the structure of the model; a property called total unimodularity.

$$\begin{aligned} &\textit{maximize} && \sum_{b\,\in B} v_b X_b \\ &\textit{subject to} && X_b - X_{\hat{b}} \leq 0 \quad \forall\, b \in B, \hat{b} \in \widehat{B_b} \\ & && 0 \leq X_b \leq 1 \qquad \forall\, b \in B \end{aligned}$$

This formulation is useful for describing the ultimate pit problem, however in 1965 Lerchs and Grossmann originally described the problem as a network problem and not with linear programming. Lerchs and Grossmann developed an algorithm for solving the ultimate pit problem which iteratively introduced necessary precedence constraints, and this was the algorithm of choice for many years after being popularized by a commercial implementation from Whittle in 1985.

There have been many different techniques developed to solve the ultimate pit problem. The floating cone algorithm from Pana (1965) is a heuristic method that does not guarantee an optimal solution but still finds use. Korobov (1974) improved on some of the floating cone algorithm's shortcomings however it also does not guarantee the optimal solution. Zhao (1992) developed a variant of the Lerchs Grossmann algorithm that guaranteed optimal results and was found to be faster, however Zhao's method was not widely adopted in industry. Mwangi (2020) presents a review of several approaches to the ultimate pit problem.

Johnson (1968) showed in detail how to express the ultimate pit problem as a network flow problem. By taking the dual of the above linear programming formulation, and performing some appropriate manipulations, an efficient construction which allows the ultimate pit problem to be solved with well-known max flow/min cut algorithms is revealed. Solving the ultimate pit problem as a network flow problem is advantageous because network flow approaches have benefitted from decades of dedicated research. Ford and Fulkerson (1962) developed much of the initial theory surrounding network flow models and provided an initial algorithm for solving them. In recent years the Push-Relabel algorithm from Goldberg and Tarjan (1988) and the Pseudoflow algorithm from Hochbaum (2001) are preferred because they are faster. These modern network flow algorithms compute precisely the same results as the venerable Lerchs and Grossmann algorithm but do not take near as long.

## 2.3 Network Terminology

A network, or graph, is a mathematical structure which models pairwise relationships between objects. Networks are used to represent things which are both abstract and concrete. Many problems become simpler when thought of through the lens of networks because turning a real world problem into a network requires one to think cogently about which of the components of the problem can be combined and represented as *nodes* and then how best to model the relationships between those components with *directed* or *undirected arcs* (Ahuja et al. 1988).

A directed arc between two nodes indicates that the arc has a specific orientation. The beginning and ending points of a directed arc are called the *tail* and *head* respectively. An undirected arc between two nodes does not have an order and only indicates that a relationship exists between the nodes.

A sequence of arcs traversed in any direction between two different nodes is a *path*. Paths are defined such that they only go through nodes and arcs at most once. If a network is constructed such that any two nodes are connected by exactly one path it is called a *tree*, an example is on the left in Figure 3. Often trees have a special node designated as the *root* from which there could be many *sub trees* or *branches*.

In the ultimate pit problem blocks are represented as nodes and precedence constraints as directed arcs. This is a special kind of network called a *directed acyclic graph*. Acyclic means that the network does not contain any directed cycles. A *closure* of a directed network is a set of nodes such that there are no arcs with their tails inside the closure and their heads outside. In the ultimate pit problem, all closures of the network are valid pits because the restriction on directed arcs ensures there are no precedence violations. An example closure is shown on the right in Figure 3. The ultimate pit, therefore, is the smallest maximum valued closure of the precedence graph.

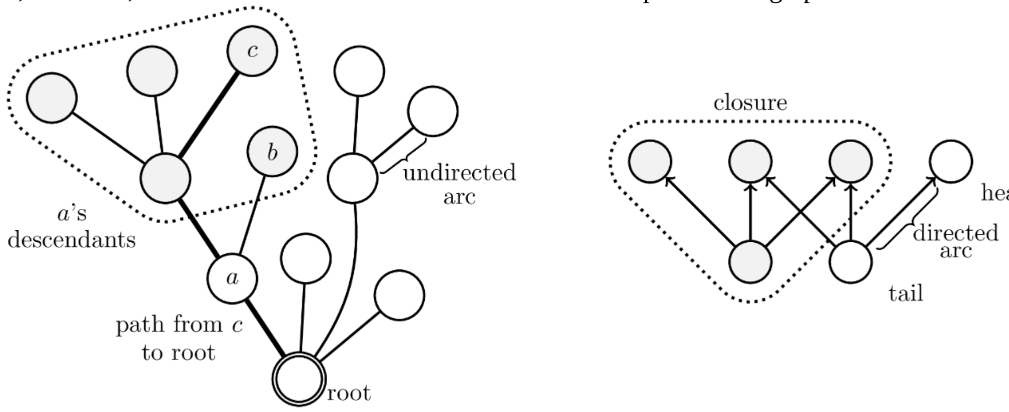

**Fig. 3** Left: A network which is a tree with associated terminology. Right: a network which is a directed acyclic graph

## 2.4 Network Flows and Cuts

Networks are used to model many different real-world problems, one of special interest to us are network flow problems. In a network flow each arc has a maximum allowed capacity and an assigned flow. Two special nodes are identified as the *source*, denoted with an $S$, and the *sink* with a $T$. Flow originates at the source node and terminates at the sink node. Usually, every other node must satisfy a *flow-balance* constraint which requires that the amount of flow into the node be equal to the amount leaving. Network models can be used to model fluids in pipes, power in an electrical grid, traffic on roads, and other similar things (Ahuja et al. 1988). A small example network flow model is shown in Figure 4. Flow is only permitted along the directed arcs, up to the capacity of the arc, from their tail to their head.

In Figure 4 the current flow from the source to the sink is four units, however it is possible to route additional flow through this network. The bolded arcs through the middle of the network can carry one additional unit of flow. And the path along the top could take an additional two units. If both paths were saturated the flow for this network would be seven, which is the *maximum flow*.

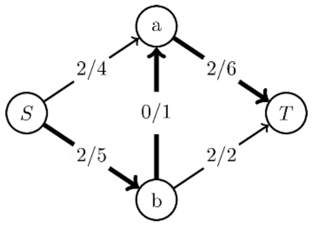

**Fig. 4** An example network flow model. Source 'S' and sink 'T' nodes are labeled. Numbers on arcs indicate 'flow' / 'capacity'. The bolded arcs show a possible augmenting path

In a network there are many ways to cut the network into two pieces. If the partitions are organized such that one side contains the source and the other side contains the sink then these two sets are called an s-t cut. In an arbitrary cut the arcs that cross from one partition to the other are said to be a part of the cut-set. However, in s-t cuts only arcs going from the source side to the sink side are included.

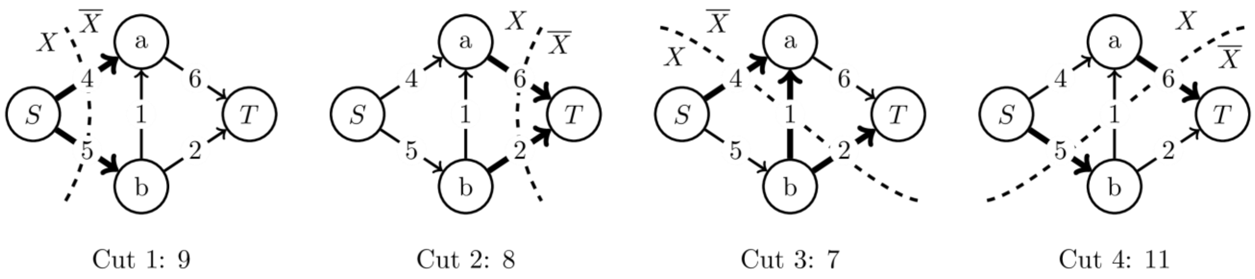

**Fig. 5** The four different possible s-t cuts for the network in Figure 4. Numbers on arcs are the arc's capacity. The *cut-set* arcs are bolded and the total cut capacity is shown below each cut

For the small network in Figure 4 there are four different possible s-t cuts as shown in Figure 5. Note that the cut-set corresponding to cut 4 only consists of two arcs despite appearing to go through three. This is because the middle arc, from $b$ to $a$, goes from the sink side to the source side.

The *capacity* of an s-t cut is the sum of the arc's capacities in its cut-set. The capacity for each possible cut in Figure 5 is given in Table 1 below.

| s-t cut | $X$ | $\bar{X}$ | *cut-set* | Capacity |
|---|---|---|---|---|
| 1 | $\{S\}$ | $\{a, b, T\}$ | $\{(S, a), (S, b)\}$ | 9 |
| 2 | $\{S, a, b\}$ | $\{T\}$ | $\{(a, T), (b, T)\}$ | 8 |
| 3 | $\{S, b\}$ | $\{a, T\}$ | $\{(S, a), (b, T)\}$ | 7 |
| 4 | $\{S, a\}$ | $\{b, T\}$ | $\{(S, b), (a, T)\}$ | 11 |

**Table 1** The source set, sink set, cut-set and capacity of the four possible cuts for the graph in Figure 4

It so happens that the maximum possible flow through a network is equal to the capacity of the minimum cut – this is formalized in the Max-Flow Min-Cut theorem. Intuitively, the arcs in the cut-set of the minimum cut correspond to the ‘bottle-neck’ of the network flow model. There is no way to fit more flow through the cut-set without increasing its capacity, and if there was some other path around the bottleneck than it would not be a valid s-t cut.

Most formal proofs of this theorem confirm the above intuition by showing that if the max flow did not equal the minimum cut there would be a contradiction. A very early discussion of this theorem appears in Ford and Fulkerson (1962).

## 2.5 Source Sink Form

Johnson (1968) showed that it is possible to formulate the ultimate pit problem as a max flow network problem where obtaining the max flow / minimum cut defines the ultimate pit. In this construction each node represents a single block of the original block model. Each node is connected by an arc from the source to the node if it has a positive economic block value, and from the node to the sink if negative. The capacity for these arcs is the absolute value of the block's economic value. Blocks with zero economic value are not connected to the source or the sink. Finally, for each precedence constraint there is an arc between the lower and higher block with infinite capacity.

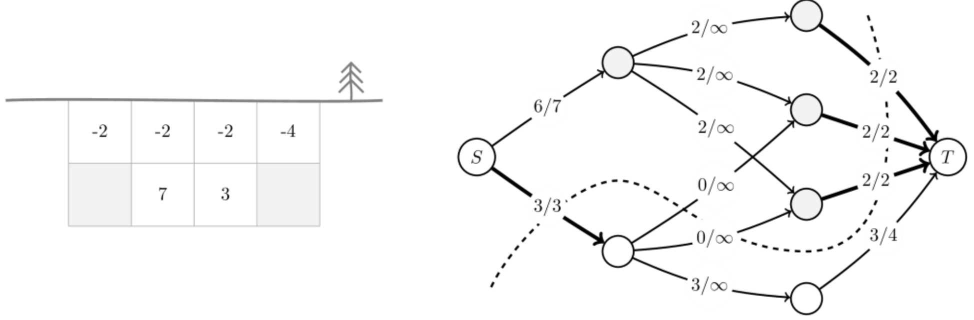

**Fig. 6** Left: An example 2D block model. Right: The associated network flow problem rotated 90° with the maximum flow and minimum cut. Numbers on arcs are flow / capacity

If this special network is solved for the max flow and minimum cut, then the blocks on the source side of the minimum cut correspond precisely to the ultimate pit limits as shown on the right in Figure 6. The capacity of the minimum cut is 9, which is equal to the maximum flow through the network. The difference between the sum of the capacities of the source adjacent arcs and the maximum flow is the ultimate pit value.

## 3 The Pseudoflow Algorithm

## 3.1 Preliminaries

In this section we describe the pseudoflow algorithm in a manner that is informative for professionals in the mining industry and therefore we will take certain theoretical liberties and specialize the algorithm so that it works only on inputs as described in Section 2.5. The generic pseudoflow algorithm must contend with additional complexities and is required to compute the final flow values on arcs; for those details the reader may review Hochbaum (2008). In solving for the ultimate pit, the final flow values are of no real interest other than to define the minimum cut. In this minor variation of the generic pseudoflow algorithm we will concern ourselves only with computing the ultimate pit as quickly as possible.

The pseudoflow algorithm is very similar to the Lerchs and Grossmann algorithm, as explained in Hochbaum (2001). Both algorithms operate iteratively by selecting a violated precedence constraint, enforcing it, and then adjusting necessary information for the next iteration. Where the pseudoflow algorithm deviates from the Lerchs and Grossmann algorithm is that it leverages the idea of flow instead of mass, it provides some machinery for selecting which precedence constraint to introduce, and maintains different structures which are easier to update efficiently.

At each iteration the pseudoflow algorithm maintains a pseudoflow on the network and a special structure called a normalized tree. A pseudoflow is a relaxed flow where nodes are not required to satisfy the flow balance constraints described in Section 2.4. Nodes which have more inflow than outflow are said to have an excess and nodes with more outflow than inflow have a deficit. On the left in Figure 7 we introduce our notation for the pseudoflow. The numbers on the arcs are the current flow on the arc. If the arc does not have a number, then the flow is zero. The numbers inside the nodes are the current excesses (when positive) and deficits (when negative). If a number is omitted inside a node then it is zero and this means that the node satisfies the flow balance constraint.

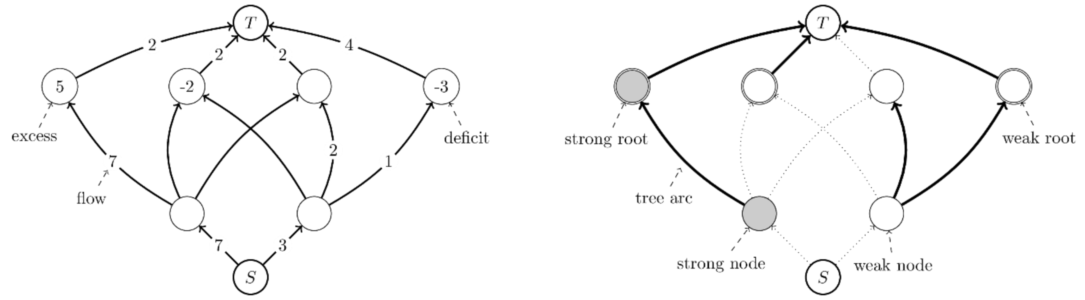

**Fig. 7** Left: The maintained pseudoflow on the flow network is notated with numbers on arcs for the flow, and numbers within nodes for excesses or deficits. Right: Thick arcs are a part of the normalized tree, and dotted arcs are not. Gray nodes are strong, white nodes are weak

In our notation we do not include the capacity on any arc. This is because the capacities of all inner arcs, representing the precedence constraints, are infinity and the flows along the source and sink adjacent arcs are always at maximum capacity.

The normalized tree is a tree in that there is exactly one unique path between any two nodes. It is also a rooted tree, meaning there is a special node, called the main root, such that every other node is a descendant of this main root. The normalized part of the name requires that only the nodes which are immediately adjacent to the main root can carry excesses or deficits. The tree remains a tree for the entire algorithm meaning that any changes to the tree require adding and dropping arcs from the tree simultaneously.

In the classical description of the pseudoflow algorithm the source and sink nodes are combined into a single node which serves as the main root. However, for our purposes we will continue to draw the source and sink nodes as two distinct circles. This does make it such that the normalized tree will appear disconnected, but it makes showing both the initialization and the progression of the pseudoflow algorithm much easier. It is important to realize that the two source and sink circles are the same main root node - just seen in two different places.

In addition to the main root we call each node that is immediately adjacent to the main root, roots. These secondary roots are classified as strong or weak based on if they have an excess or a deficit in the current pseudoflow respectively. Recall that for the tree to be normalized these secondary roots are the only nodes permitted to have excesses or deficits. We also say that any node which is a descendant of a strong root is strong, and similarly any node which is a descendant of a weak root is weak.

Figure 7 introduces our notation for the normalized tree on the right. Line weight and style for arcs indicates membership in the normalized tree, secondary roots are in double circles, and the color of the node indicates strength. Gray nodes are the current strong nodes and white nodes are weak.

These two pieces of notation are then superimposed on top of one another during the execution of the algorithm. There are a few important points to remember; the algorithm starts with a normalized tree and maintains that structure throughout, and the source and sink circles, labeled S and T, correspond to the same single main root node. When we discuss a path from a node back to the main root - it may go to either the source or the sink circle.

## 3.2 Initialization

The first step in the pseudoflow algorithm is to construct an initial normalized tree and an initial pseudoflow. It is possible to start the pseudoflow algorithm from any normalized tree with a valid pseudoflow, but the simplest starting point is to fully saturate all source and sink adjacent arcs, and include all source and sink adjacent arcs in the normalized tree as in Figure 8.

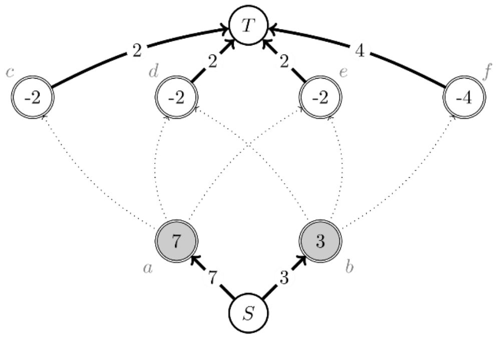

**Fig 8** The initial normalized tree, letters near nodes are node names and not a part of the notation

## 3.3 Algorithm Steps

At each iteration of the pseudoflow algorithm an arc corresponding to a precedence constraint between a strong node and a weak node is selected and chosen as the *merger* arc. This merger arc is then introduced into the normalized tree and the arc between the strong root and main root is removed. The pseudoflow is then adjusted so that the tree remains normalized. This requires adjusting flows along the path from the strong root to the weak root. During this re-normalization process there may be tree arcs which must be removed from the normalized tree which results in new branches. When there are no more precedence arcs between strong and weak nodes the algorithm stops, and all surviving strong nodes are the ultimate pit.

In practice choosing which merger arc to use has a large impact on the performance of the algorithm. The pseudoflow algorithm uses a specific labeling scheme, described in Section 3.4, to guarantee good performance and avoid the sort of problems which plague the Lerchs and Grossmann algorithm. In this initial treatment of the pseudoflow algorithm we forego the use of labels and follow the sequence of merger arcs as indicated on the left in Figure 9 beginning with the precedence arc between nodes $b$ and $e$. This sequence is *not* a good sequence, but we use this sequence because it forces a split.

Once the appropriate merger arc is chosen the algorithm identifies the corresponding strong root and the weak root. The strong root will, by definition, have some positive excess. Proceed as follows:

1. Update the normalized tree by:
    a. removing the tree arc connected to the strong root, and
    b. adding the merger arc.
2. Set a variable *delta* to be the excess of the strong root.
3. 'Push' *delta* units of flow along the path from the strong root to the week root. Update the excesses / deficits / strength as you go, and for each arc along the path:
    a. If the arc is directed in line with the flow: Increase the flow on the arc by *delta*.
    b. Else, try to decrease the flow on the arc by *delta*
        i. If the flow is strictly greater than *delta*: Set the flow to the current flow less *delta*
        ii. Else: Split flow on this arc (see below for details on splitting).

The result of the first merge, between $b$ and $e$, is shown in the middle in Figure 9. For the second merge, between $b$ and $f$, the strong root is $e$ and the weak root is $f$. The one unit of excess at $e$ is pushed along the path but does not exceed the deficit at $f$ so the entire branch is reclassified as weak, with a deficit of 3.

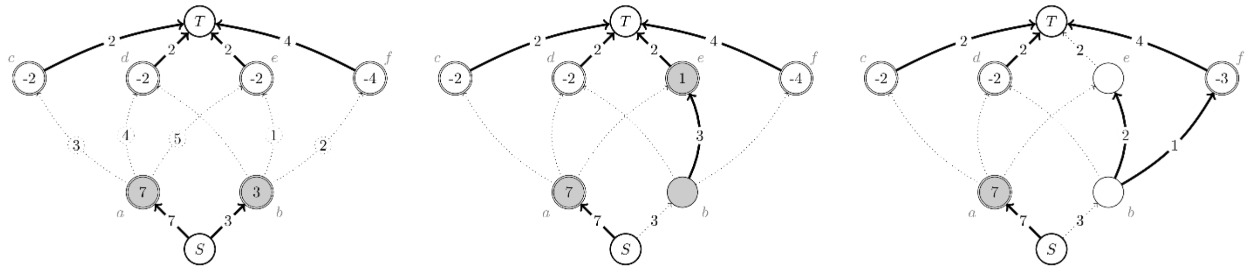

**Fig. 9** Left: The starting network, circled numbers on nodes indicate the order of merging arcs in this example. Middle: The result of the first merge between nodes b and e. Right: The result of the second merge between nodes b and f

In Figure 10 the left and middle graphs show the result of the 3rd and 4th merges between $a$ and $c$, and $a$ and $d$. For the final merge between $a$ and $e$ we first update the tree and begin pushing $d$'s excess along the path towards the root. We reduce the flow between $a$ and $d$ by 3, increase the flow between $a$ and $e$ by 3 but now we have a problem. We would like to decrease the flow between $b$ and $e$ by 3 (our current value for delta), but this would lead to a negative flow which is not allowed. We can, however, reduce the flow to zero which will leave one unit of excess flow on $e$. Then we split on this arc as follows:

1. Update *delta* to be the flow along the splitting arc, push the new delta units along the splitting arc which will reduce the split arc flow to zero and leave positive excess at the head node

2. Update the normalized tree by:
    a. removing the split arc, and
    b. adding the arc from the head node to the root
3. Continue pushing the new *delta* units of flow along the remaining path to the weak root

There may be several of these splitting operations in a single merge operation with a longer path between roots.

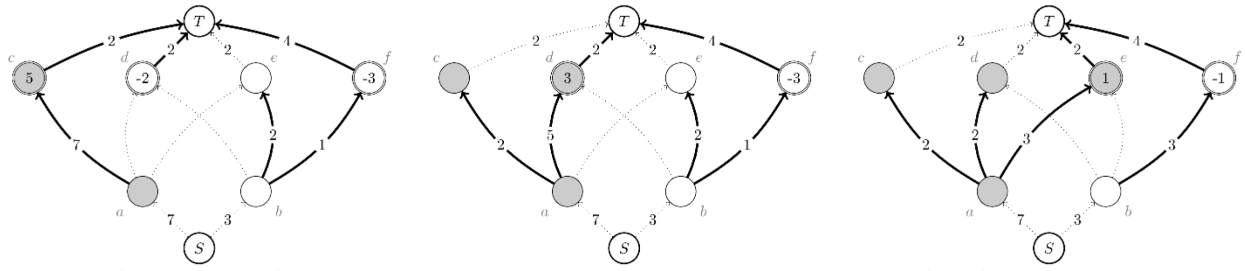

**Fig. 10** Left: The result of the third merge between nodes a and c Middle: The result of the fourth merge between nodes a and d. Right: The result of the final merge between nodes a and e, which requires splitting on the arc between b and e

When there are no longer any precedence arcs between strong nodes and weak nodes the algorithm terminates. Any nodes which are still classified as strong at the end of the algorithm are then the ultimate pit, and the sum of their excesses is the ultimate pit value.

## 3.4 Labeling

Hochbaum introduces a labeling scheme to improve the performance of the pseudoflow algorithm. Intuitively, the labeling scheme forces the algorithm to carefully choose the merger arc at each iteration; avoiding certain sequences of merger arcs which would require more iterations than necessary. Specifically, the labeling scheme makes it such that the merger arc between nodes s and $w$ cannot be used again until the labels on each $s$ and $w$ have increased by at least one.

All nodes are initially assigned a label of 1. When selecting a merger arc between a strong node $s$ and a weak node $w$ the algorithm *must* select an arc such that the label of $w$ is as low as possible. Once the merger arc is selected, the label of *all* strong nodes is set to the maximum of its current label and $l_w + 1$. These straightforward steps are enough to improve both the theoretical complexity of the pseudoflow algorithm and the practical performance.

To illustrate the labeling procedure we adjust the value of block $c$ from -2 to -4 and modify the sequence of merging operations. Initially all blocks have a label of 1. However, as soon as the first merger arc between $a$ and $e$ is selected the labels of $a$ and $b$ are set to 2. The state of the network after the first three merges is shown on the right of Figure 11. Blocks $a, b, d$ and $e$ all have a label of 2. At this point, due to the labeling scheme, the only allowed merger arc is between $b$ and $f$ – because $f$ has the lowest label. This merge will lead to the immediate termination of the algorithm as all blocks will become weak. If the labeling scheme were *not* used then we could choose the next merger arc to be between $b$ and $e$ (or $b$ and $d$). Either of these choices would require additional iterations.

The labeling scheme presented here follows Hochbaum (2008) and specifies a relabeling operation to occur across all strong nodes after a merger arc is selected. This is not strictly necessary. Chandran and Hochbaum (2009) give advice on a modification to the labeling scheme that delays relabeling and improves the computational efficiency. Our implementation uses the more performant delayed labeling scheme which is also present in Hochbaum's implementation.

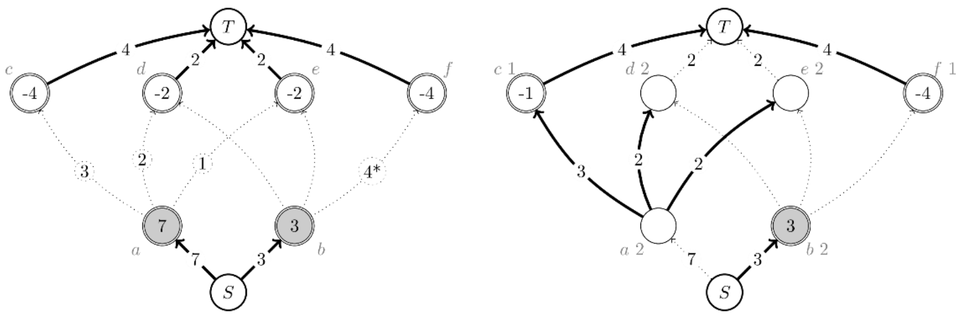

**Fig. 11** Left: The slightly modified example with a different sequence of merger arcs; as circled numbers on arcs. Right: The pseudoflow network after the first three merging operations. Labels are given as numbers next to the node names. a,b,d,e all have label 2. c andf have a label of 1. The only valid merging operation is b to f

## 4 MineFlow

### 4.1 Implementation Details

Hochbaum's implementation of the pseudoflow algorithm must contend with complexities which are not present in the ultimate pit problem. An entire class of potential splitting operations is precluded because all precedence arcs have infinite capacity, and all arcs are oriented from the source node towards the sink node. Additionally, we do not need to store the capacity on arcs nor the information necessary to recover the maximum flow after identifying the minimum cut. We take advantage of these differences in our implementation of the pseudoflow algorithm.

MineFlow is a C++17 library which exposes a few straightforward classes for defining precedence graphs and computing ultimate pit limits with pseudoflow. The pseudoflow solver itself requires the integral block values, and a class which can be queried for the precedence constraints. It is important that we do not store the entire precedence graph in memory at any time: instead it is defined implicitly by using precedence patterns or lazy evaluation when possible. A typical large block model for the ultimate pit problem may have billions of precedence constraints, but only a very small fraction of them will be used when calculating the ultimate pit, we can save a great deal of time by not considering them unless strictly necessary.

Economic block values must be provided to MineFlow as integer values. MineFlow can support arbitrary fixed sized integers. 256-bit integers have been tested with a minor decrease in performance. In practice economic block values can be scaled and rounded as necessary.

To use the library with user specified data and precedence constraints it is necessary to define two classes with four virtual methods. A class inheriting from `IBlockValues` must be defined with two methods: `NumBlocks` and `BlockValue` which return the number of distinct blocks and their integral economic block values respectively when called. Additionally, a class inheriting from `IPrecedenceConstraints` must be defined which implements `NumBlocks` and `Antecedents`. The `Antecedents` function must return an iterator for each block that specifies which blocks must be mined first. There are examples of both of these classes included in the repository.

In addition to the C++17 library, MineFlow includes a standalone executable which operates on flat text files. This executable is naturally much less flexible than the library but covers functionality for large regular block models and smaller explicit datasets. Both the library and the executable are released completely open source with no dependencies on third party libraries. This makes MineFlow a useful library for both academic and commercial applications.

The executable has a straightforward command line interface to define common types of precedence constraints, and a basic parameter file to specify minimum search precedence patterns that allow precedence patterns to be defined with a custom block size and pit slopes that vary by direction. The executable also supports solving custom problems with each block's value and precedence constraints defined in two files. Complete up-to-date documentation of the executable and library may be found alongside the solver in the main repository.

### 4.2 Algorithm Comparison

To verify that our implementation is correct and computationally efficient we collected 5 block models and imported them into several different commercial packages. We also transformed the problem to the pure max flow format to compare with Hochbaum's original implementation. In all cases MineFlow computed the ultimate pits in less time than the other methods.

For each package and each problem 5 runs were completed. The times presented Table 2 are given as the average of the middle 3 times, excluding the fastest and the slowest. All computations were completed on a Windows 10 machine with a 3.70GHz Intel Xeon XPU E3-1245 v6 processor and 32.0 GB of RAM. The times were measured as the complete time as reported by the package. This includes the time to read the input and write the output which can vary between packages.

| Dataset | | | Solution time (sec), average middle 3 of 5 runs | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | Blocks (Mined / Total)* | Precedence* | Package A | Package B | Package C | Package D | Package E | Hochbaum's `pseudo_fifo` | MineFlow |
| Bauxite** | 74,412 / 374,400 | 5,349,104 | 25 | 22 | 4 | 9 | 9 | 3 | 0 |
| Copper Case Study | 357,304 / 1,827,500 | 28,321,632 | 179 | 99 | 14 | 85 | 43 | 15 | 2 |
| Copper Pipe | 198,078 / 2,754,000 | 43,877,152 | 399 | 183 | 64 | 290 | 109 | 24 | 3 |
| McLaughlin | 345,936 / 2,817,920 | 44,495,400 | 243 | 168 | 37 | 34 | 52 | 24 | 3 |
| Gold Vein | 602,150 / 16,244,739 | 264,007,172 | 721 | 764 | 169 | 752 | 556 | 81** | 9 |

**Table 2** The solution times obtained when solving for the ultimate pit for the 5 different datasets. Times are given in seconds as the average middle 3 of 5 runs.
* The precise number of mined blocks and the total pit value varies by less than 1% between packages due to different implementations of precedence constraints.
** This Ultimate Pit is shown in Figure 1.
*** This process ran out of available memory during the initialization step, an alternate computer with more available RAM was used to obtain this time.

The commercial packages are anonymized to respect the wishes of some of the software vendors. All five packages use a network flow based implementation however the precise details are not published or publicly available. One package uses the Push-Relabel algorithm and the other four use some variant of Pseudoflow.

Each commercial package computed near identical results with the only changes being due to minor departures in how each package handles precedence constraints. MineFlow and `pseudo_fifo` computed identical results for all datasets as `pseudo_fifo` is a general utility with no means of generating precedence constraints by itself; MineFlow was used to create the necessary input files. Notably package C also computed identical results to MineFlow for all datasets. Departures due to precedence constraints were less than 1% in both overall pit value and the number of blocks mined.

## 5 Conclusions

The ultimate pit problem remains a relevant subproblem in modern open pit mine planning. Circular analysis, scheduling, uncertainty management, and other processes rely on computationally efficient and available ultimate pit solvers. Additionally, the Bienstock Zuckerberg algorithm, Bienstock & Zuckerberg (2009), uses the ultimate pit problem as a subroutine which is repeatedly solved to solve the precedence constrained production scheduling problem. Certain open pit mine planning problems that can be formulated as the ultimate pit problem with side constraints can be approached with Lagrangian relaxation and repeated applications of MineFlow. . It is common to solve ultimate pit problems thousands of times in these approaches and MineFlow makes that much faster.

Commercial packages have adopted the pseudoflow algorithm, or similar max flow algorithms, for solving the ultimate pit problem, however implementations vary in terms of availability, flexibility, and computational efficiency. Our own need for MineFlow arose when approaching large open pit mine planning problems that used the ultimate pit problem as a subroutine. In developing MineFlow we found room for optimizations and modifications which substantially improve the practical performance of the algorithm. Implicit, or lazy, precedence constraints, precluding impossible splitting operations, and eschewing the flow recovery step and associated bookkeeping have led to a computationally efficient ultimate pit solver which outperforms current commercial implementations.

Additionally, we developed a compact notation for the pseudoflow algorithm with the aim of allowing future generations of mining engineers and technologists to learn network flow techniques which are the current best practice in ultimate pit optimization.

## References

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1988). Network flows: Theory, algorithms, and applications. Pearson

Bienstock, D., & Zuckerberg, M. (2009). A new LP algorithm for precedence constrained production scheduling. In Optimization Online (pp. 1–33)

Caccetta, L., & Giannini, L. M. (1988). The generation of minimum search patterns in the optimum design of open pit mines. In AusIMM Bull Proc (Vol. 293, No. 5, pp. 57–61)

Chandran, B. G., & D. S. Hochbaum. (2009) A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem". Operations Research 57

Chen, T. (1976). 3D pit design with variable wall slope capabilities. In 14th symposium on the application of computers and operations research in the mineral industries (APCOM)

Darling, P. (Ed.). (2011). SME mining engineering handbook. In Society for mining, metallurgy, and exploration (Vol. 1)

Deutsch, M., E. González, & M. Williams. (2015) "Using simulation to quantify uncertainty in ultimate-pit limits and inform infrastructure placement". Mining Engineering 67

Díaz, A. B. et al. (2021) "Calculating ultimate pit limits and determining pushbacks in open-pit mining projects". Resources Policy 72

Ford, L. R., & D. R. Fulkerson. (1962) "Flows in networks". Princeton University Press

Gilani, S. O., & J. Sattarvand. (2015) "A new heuristic non-linear approach for modeling the variable slope angles in open pit mine planning algorithms". Acta Montanistica Slovaca 20

Goldberg, A. V., & R. E. Tarjan. (1988) "A new approach to the maximum-flow problem". Journal of the ACM (JACM) 35

Hochbaum, D. S. (2001) "A new—Old algorithm for minimum-cut and maximum-flow in closure graphs". Networks: an International Journal 37

---. (2008) "The pseudoflow algorithm: A new algorithm for the maximum-flow problem". Operations Research 56

Johnson, T. B. (1968). "Optimum open pit mine production scheduling" (No. ORC-68–11). California Univ Berkeley Operations Research Center

Khalokakaie, R., P. A. Dowd, & R. J. Fowell (2000). "Lerchs-Grossmann algorithm with variable slope angles". Mining Technology 109

Korobov, S. (1974). Method for determining optimal open pit limits. Ecole Polytechnique de l'Université de Montréal. Technical report EP74

Lerchs, H., I. Grossmann. (1965) "Optimum design of open-pit mines". Operations Research 12

Mwangi, A. D. et al. (2020) "Ultimate pit limit optimization methods in open pit mines: A review". Journal of Mining Science 56

Pana, M. T. (1965). The simulation approach to open pit design. In 5th symposium on the application of computers and operations research in the mineral industries (APCOM) (pp. 127–138)

Zhao, Y. (1992). Algorithms for optimum design and planning of open-pit mines. The University of Arizona

**Citation**

```
@article{deutsch2022mineflow,
  author={Deutsch, Matthew and Da{\u{g}}delen, Kadri and Johnson, Thys},
  title={An Open-Source Program for Efficiently Computing Ultimate Pit Limits: MineFlow},
  journal={Natural Resources Research},
  year={2022},
  month={Mar},
  day={17},
  issn={1573-8981},
  doi={10.1007/s11053-022-10035-w},
  url={https://doi.org/10.1007/s11053-022-10035-w}
}
```

Deutsch, M., Dağdelen, K. & Johnson, T. An Open-Source Program for Efficiently Computing Ultimate Pit Limits: MineFlow. Natural Resources Research (2022). https://doi.org/10.1007/s11053-022-10035-w