# Towards Solving Large-Scale Precedence Constrained Production Scheduling Problems in Mining

Angus Kenny
School of Science (CSIT), RMIT University
Melbourne, Australia
angus.kenny@rmit.edu.au

Xiaodong Li
School of Science (CSIT), RMIT University
Melbourne, Australia
xiaodong.li@rmit.edu.au

Andreas T. Ernst
School of Mathematical Sciences, Monash University
Melbourne, Australia
andreas.ernst@monash.edu

Dhananjay Thiruvady
School of Mathematical Sciences, Monash University
Melbourne, Australia
dhananjay.thiruvady@monash.edu

## ABSTRACT

Pit planning and long-term production scheduling are important tasks within the mining industry. This is a great opportunity for optimisation techniques, as the scale of a lot of mining operations means that a small percentage increase in efficiency can translate to millions of dollars in profit. The precedence constrained production scheduling problem (PCPSP) combines both of these aspects of mine optimisation and aims to find a solution which tells a mining company what part of the orebody to mine, and at what time during the life of the mine. This paper presents a GRASP-Mixed Integer Programming hybrid metaheuristic algorithm for solving the PCPSP which consists of two parts: a fast, period-by-period, random construction phase and a local improvement heuristic. It is compared to the current published state-of-the-art results on well known benchmark problems from *minelib* [5] and is shown to give better quality results in four of the six instances, and within 2% of the LP upper bound in the remaining two. The PCPSP is a good candidate for hybrid metaheuristics as the size of the problems make solving them with mathematical solvers alone intractable.

## CCS CONCEPTS

**•Mathematics of computing → Combinatorial optimization;**
**•Applied computing → Industry and manufacturing;**

## KEYWORDS

Applied computing, Hybrid algorithms, Mixed integer programming, Mine planning

**ACM Reference format:**
Angus Kenny, Xiaodong Li, Andreas T. Ernst, and Dhananjay Thiruvady. 2017. Towards Solving Large-Scale Precedence Constrained Production Scheduling Problems in Mining. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071241


*GECCO '17, Berlin, Germany*

DOI: http://dx.doi.org/10.1145/3071178.3071241

## 1 INTRODUCTION

Open-pit mining is a very important industry in Australia and around the world. Two of the most critical tasks within the lifecycle of an open-pit mine is planning and production scheduling. These tasks allow the mine operator to estimate the total value of the mine over its life and also to identify areas for excavation that will yield the most value. Proper planning of a mine ensures maximum profit for the operator and because this is typically talked about in the hundreds of millions of dollars, it is an excellent application for optimisation techniques as very small changes in efficiency can still translate to significant sums of money [10].

In order to make this problem solvable by combinatorial methods, the earth to be mined (known as the *orebody*) is typically discretised into a three-dimensional array of *blocks* with a given value based on the ore content and the cost required to excavate it. These values are calculated by taking core samples and using geological and statistical methods to estimate the value of each block.

Problems in mine planning and production scheduling are very large and tend to have few side-constraints (often well under a hundred), but many blocks and many, many more precedence constraints governing when blocks can be mined [1]. This means traditional mathematical solvers are unable to solve these problems without first using some form of decomposition, making these problems perfect candidates for metaheuristics and metaheuristic-MIP hybrids, despite there being very little in the literature.

Due to the sensitive nature of information surrounding mining enterprises, obtaining problem data for academic research can be challenging, however the website *minelib* [5] has a repository of problems and results that are freely available for the general public. These sets contain data for versions of the problem such as the ultimate pit limit (UPIT), constrained pit limit (CPIT) and the precedence constrained production scheduling problem (PCPSP). It is this latter set that will be considered here.

This paper presents a Greedy Randomised Adaptive Search Procedure (GRASP)-Mixed Integer Programming (MIP) hybrid algorithm that is shown to be better than the current state-of-the-art in four of six of the *minelib* instances, and within 2% of the linear programming (LP) upper bound for the remaining two. It uses a period-by-period decomposition technique to obtain a population of fast, feasible solutions of reasonable quality before employing a local improvement heuristic with a sliding window mechanic that

considers smaller parts of the larger problem individually. Due to the size of the search space and complexity of the constraints, applying sophisticated evolutionary methods may not be straightforward; Singh et. al [14] suggested that simpler heuristics worked better than genetic algorithms on a similarly large-scale mining problem.

This paper is organised as follows: Section 2 introduces this interesting and unique problem, gives a mathematical formulation and discusses how the problem is modelled and gives some of the techniques currently used to solve various forms of the problem. Section 3 discusses the period-by-period formulation, developed to provide a way of decomposing the problem into manageable parts. Section 4 presents the proposed algorithm and discusses its various aspects and attributes. Section 5 details the experiments performed and gives descriptions of the datasets used from *minelib*. Section 6 presents and discusses the results obtained from the experiments in Section 5; it also gives a discussion of the various parameters that are used to control the algorithm. Finally, Section 7 brings together the conclusions drawn from the results and outlines some directions for potential future research.

## 2 BACKGROUND

This section gives a description of the problem, introduces ways in which it can be modelled and gives a brief survey of how other researchers have approached solving it.

### 2.1 Problem description

To make planning the mine easier, the orebody is divided up into discrete blocks and each one given a value based on the percentage of ore and the cost to mine it (Figure 1). These values are typically calculated using deep core drilling samples at certain points, however as the values are given in the data sets, exactly how the values are arrived at is not relevant here.

| | | | | |
|---|---|---|---|---|
| 1 | -2 | -2 | -2 | -2 |
| | 5 | 6 | -3 | |
| | | 4 | | |

**Figure 1: Vertical cross-section of an orebody model with value of blocks[1].**

Typically, there are two destinations in the given data sets: mining a block and sending it to be processed; and mining a block and sending it to a waste stockpile. The values shown in Figure 1 are the maximum of the profit from mining and processing a block and the cost of just mining it.

*Precedence constraints.* In open cut mines a block cannot be mined unless the block directly above it is also mined. This property is known as a precedence constraint and can be represented in the structure of the problem by creating a graph where each block is a node and the precedence constraints are arcs pointing from a block to block below that can be mined after it (Figure 2).

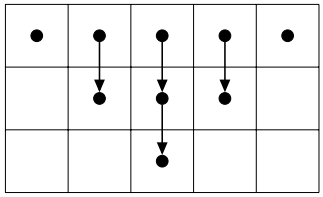

**Figure 2: Immediate precedence constraints indicate that each block cannot be mined before the one above.**

Using immediate precedence constraints alone would allow for narrow, vertical shafts to be dug in the mine. In order to let the digging and extraction equipment to be brought in, the walls of the mine have to be dug with a certain slope angle. This means that a block may not be mined, unless the block directly above it *and* the blocks all around that block are mined. These extra precedence constraints can also be encoded into the problem structure by adding extra arcs from the block to all of its successors. A 2-dimensional example is given in Figure 3; however, in practice, each (non-edge) block will have at least 9 predecessors.

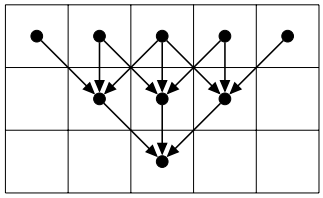

**Figure 3: Precedence graph constructed from Figure 1 with both immediate precedence and slope constraints.**

*Resource constraints.* Along with precedence constraints, there are also resource constraints which control how many blocks can be mined and sent to a certain destination. To this end, there are resource limits associated with each resource/destination. These resource limits effectively control how much mining can be done within a certain time period. The resources do not accumulate over time periods - if a particular resource is not exhausted in the current time period, it will not continue to be available in the next period. The constraints would be typically derived from the available capacity of the bottleneck operation. For example this might be the available machine hours of the dragline excavators in the pit.

*Discount rate.* A discount rate is applied to the profit obtained by mining a particular block at a particular time. This means that the later a block is mined, the less profit it generates from processing. Conversely, the later a block is mined, the smaller loss it will make from mining it if it is negatively valued.

The objective of the problem is to maximise the *net present value* (NPV) of the mine, taking into account the following restrictions:

(1) Each block is mined at most once.

[1]Figures 1, 4 and 5 in this section are adapted from an example given in the survey paper by Meagher et. al. [10].

(2) The predecessor blocks must be mined in the same period or earlier.
(3) Each block must be sent (possibly fractionally) to the destinations in the period in which it is mined.
(4) The resource limits consumed in sending blocks to their destinations must not be violated.

It should be made clear that this is a significantly simplified problem but one that captures the essential characteristics of this challenging optimisation problem. In real life, additional considerations include grade targeting (ensuring a consistent product in terms of impurities and other characteristics), more complex constraints on pit shape from operational constraints (push-backs), uncertainty in block composition and other mine or business specific considerations.

## 2.2 Modelling the problem

*Mathematical formulation.* The data for this problem consists of the following sets and parameters:

$B$ the set of blocks;
$T$ the set of time periods;
$D$ the set of destinations. Typically, $|D| = 2$;
$R$ the set of resources. Typically, $|R| = 2$;
$\mathcal{P}$ the set of precedences. $a \rightarrow b$ if $(a, b) \in \mathcal{P}$ means block $a$ must be mined before block $b$;
$p_{bdt}$ the profit for sending block $b$ to destination $d$ at time $t$ (can be negative if it is a cost only). For the minelib data this is simply $\frac{p_{bd}}{(1+\text{discount})^t}$ for some base cost $p_{bd}$ and a discount value that is typically in $(0, 1)$;
$q_{bdr}$ the amount of resource $r$ required by block $b$ if sent to destination $d$; and,
$\overline{R}_{rt}$ the amount of resource $r$ available in time period $t$.

The variables used to solve this problem are:

$x_{bt}$ is a binary variable that is one if block $b$ is mined in period $t$ or earlier (that is, the block has been removed by the end of period $t$)

$y_{bdt}$ the fraction of block $b$ sent to destination $d$ at time $t$

Using this notation, the problem is written as:

$$\max \quad \sum_{b\in B}\sum_{d\in D}\sum_{t\in t} p_{bdt}y_{bdt}, \tag{1}$$

s.t.,

$$x_{bt} \le x_{at} \qquad \forall\,(a,b)\in\mathcal{P},\ t\in T, \tag{2}$$

$$x_{bt} \le x_{b,t+1} \qquad \forall\, b\in B,\ t\in T, \tag{3}$$

$$\sum_{d\in D} y_{bdt} = x_{bt} - x_{b,t-1} \qquad \forall\, b\in B,\ t\in T, \tag{4}$$

$$\sum_{b\in B}\sum_{d\in D} q_{bdr}y_{bdt} \le \overline{R}_{rt} \qquad \forall\, r\in R,\ t\in T, \tag{5}$$

$$x_{bt}\in\{0,1\}, \quad y_{bd}\ge 0 \qquad \forall\, b\in B,\ d\in D,\ t\in T. \tag{6}$$

Note: for correctness, in (4) for the first time period, no previous $x$ is to be subtracted.

In the above formulation, constraint (2) says that block $a$ must be mined before block $b$ and represents the precedence and slope constraints; (3) states that if block $b$ is mined at time $t$, it must also be mined at time $t + 1$; (4) ensures that, although a fraction of a block can be sent to different destinations, the entire block must be either mined or not mined within a single time period; (5) are the resource constraints; and (6) says that $x_{bt}$ must be integral, but $y_{bd}$ may be fractional.

*Network flow approach.* By relaxing constraints (5) and (4), the problem then becomes a matter of finding the set of blocks that are worth mining at all throughout the whole process. This is known as the UPIT problem, and represents an upper bound on the constrained problem.

The structure of the problem dictates that the solution to the UPIT problem is also the maximum closure of the precedence constraint graph where the values of the blocks are the weight of the graph nodes. A closure of a graph is a partitioning of that graph into two sets such that all arcs between nodes in both sets are all of the same direction; either into, or out of the closure. A maximum closure is obtained when the value of the nodes in the closure is the maximum possible value. The blocks that are to be included in the ultimate pit all fall inside the maximum closure and those that will never be mined are outside.

Picard [12] showed how to find the maximum closure of a graph using maximum network flow algorithms, and these algorithms can be used to solve the UPIT problem quite efficiently. The existing arcs in the precedence graph are given infinte capacity - as they are only there to preserve predence and slope constraints - and "dummy" nodes representing the source and sink are added with arcs from the source node to each negatively valued block, and arcs from each positively valued block to the sink (Figure 4).

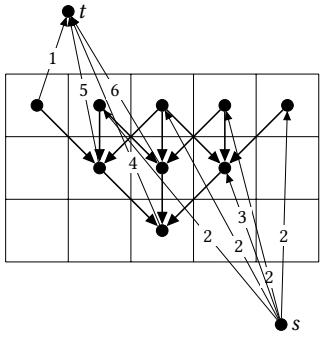


**Figure 4: Network flow graph constructed from Figures 1 and 3.**

The problem can then be solved as a normal network flow problem, the closure containing any blocks with unsaturated residual flow arcs to the sink and their predecessors (Figure 5).

The way this technique works is to "push" losses from negative valued blocks, down to higher valued blocks in order to balance these losses out. If, after these losses have been balanced out, there is still value (flow) remaining in the positively valued block (its arc is still unsaturated), this indicates that it is worth mining all of the blocks above in order to reach it.

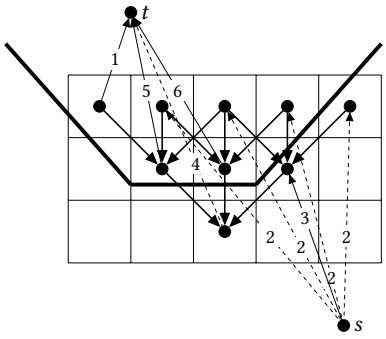


**Figure 5: Solving the network flow problem gives the maximum closure. Dashed arrows indicate saturated flow arcs and the bold line indicates the maximum closure.**

This aproach can also be used to solve sub-problems anywhere a partition of the graph is needed. Empirically, the best algorithms found to solve these maximum closure problems are the Boykov-Kolmogorov [3] and the CPLEX network simplex algorithms.

## 2.3 Related work

Despite the problem having been formalised as early as the 1960s, due to limits in computing power most of the earlier work focuses on solving the UPIT or CPIT problems [4, 9]; for a good survey of research in these related areas, see [7, 10]. Another area of related research is solving the LP relaxation, a modified version of the algorithm from Bienstock and Zuckerberg's paper [1] is used to compute the LP upper bound on *minelib*. There are a few papers that focus on solving the PCPSP as used in this research and this section outlines some of the methods used.

In order to alleviate the problems caused by the enormous number of blocks in a problem (see Table 1), Jélevez et. al. [8] uses an aggregation heuristic to group blocks into larger blocks that will all be selected or not at the same time. This method also uses a separate disaggregation heuristic to ungroup blocks when a more fine-grained search is required.

Bley et. al. [2] present a strengthened MIP formulation for the PCPSP. They include additional constraints derived by combining precedence and production constraints. In their experiments, they find that these additional constraints serve to reduce the computation time needed to solve their custom instances using CPLEX. They did not use their techniques on the *minelib* instances, likely because they are too large for CPLEX to solve.

The masters thesis of Gonzalo Muñoz [11] provides the results that are reported on *minelib* and used for comparison in this paper. This algorithm uses a modified version of the Bienstock and Zuckerberg algorithm to solve the LP relaxation and then use a topological sorting algorithm, similar to that used in [4] to construct a feasible solution from this relaxation.

# 3 SINGLE PERIOD PCPSP

Modelling the fully constrained problem as a network flow requires significant expansion of the problem graph. Figure 6 shows that each destination requires an extra block node and arc; and each time period requires an extra copy of the full problem graph. This is because the model in Equations 1-6 requires decision variables ranging over blocks, destinations *and* time periods. Clearly this expansion makes large instances of the problem intractable for most algorithms, so some kind of decomposition technique is required.

## 3.1 Reduced formulation

There is nothing that can be done about the increase in problem size that comes from adding destinations, as the destinations are inextricably linked to the resource constraints; and the increase of the size of the problem is only $BD$, where $B$ is the number of blocks and $D$ the number of destinations. However, significant reductions in problem size can be made by removing any reference to time from the formulation and solving the problem one time period at a time. Removing time considerations from equations (1) to (6) gives:

$$\max \quad \sum_{b\in B}\sum_{d\in D} p_{bd}y_{bd}, \tag{7}$$

s.t.,

$$x_b \le x_a \qquad \forall\ (a,b)\in\mathcal{P}, \tag{8}$$

$$\sum_{d\in D} y_{bd} = x_b \qquad \forall\ b\in B, \tag{9}$$

$$\sum_{b\in B}\sum_{d\in D} q_{bdr}y_{bd} \le \overline{R}_r \qquad \forall\ r\in R, \tag{10}$$

$$x_b\in\{0,1\} \quad y_{bd}\ge 0 \qquad \forall\ b\in B, d\in D. \tag{11}$$

This, much simpler, formulation can then be solved iteratively for each period, using any MIP solver such as CPLEX, by fixing the blocks mined in each previous period, until all blocks in the ultimate pit have been mined.

# 4 GRASP HEURISTIC

This Section outlines the proposed greedy randomised adaptive search procedure (GRASP) heuristic named *minePS+LI*. As with most GRASP heuristics, it consists of two phases; a randomised solution construction phase and a local improvement phase [13].

## 4.1 Random solution construction

Deterministic solutions to the problem can be greedily constructed using the basic single-period CPLEX-based model discussed previously. However, in order to be considered a GRASP heuristic, a method of generating randomised solutions is required.

The CPLEX model works by finding the optimal subset of blocks to mine in that period from a superset of blocks that are included in the original model. By default, this superset is the entire set of blocks available in the UPIT solution but there are a number of drawbacks to this method. Firstly, due to resource constraints, not every block in the UPIT solution is reachable in every period; this means that there are a significant number of blocks included in the model for a given period which will *never* be mined because there is just not enough resources to reach them. Secondly, as problem size increases, the inclusion of every reachable block in the model still results in very large models; and therefore significantly slower solution times. Finally, including the same blocks in the model every time results in the same (or extremely similar, due to CPLEX's

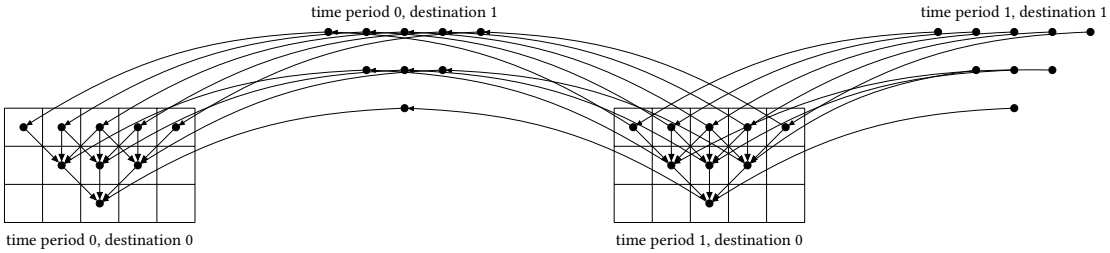


**Figure 6: Time and destination expanded problem graph for two time periods and two destinations. An extra node and arc is added for each destination, and a full copy of the destination expanded graph is added for each time period.**

random processes) solutions being generated. All three of these issues can be addressed by *random cone selection*.

In order to excavate a block, all of its predecessor blocks must first be excavated. These predecessors take the shape of a cone, which starts with its vertex at the block in question and extends upward and outward until it reaches the surface. The total value of these cones can be calculated by summing up the maximum profit for each of the blocks contained within the cones and this information can be used to determine which blocks are included in the model. The minimum resources required to excavate a cone can also be computed while calculating the value of the cone, and this information can be used to determine whether a block is reachable in a given period and therefore if it should be included.

Random cone selection works by computing all such cones in the orebody model and then selecting with some probability $p$ (typically, 0.5), cones which consume $\rho$ fraction of the resources available for the current period and whose base block is not a member of another cone. The cones are first sorted by value, in order to bias the algorithm towards selecting high-valued cones, and selected by adding all blocks from the current cone into the model. This continues until some multiple $\mu$ of the resource limit for the current period is exceeded after adding all blocks in a selected cone. A full description of these parameters is given in Section 6.

Having populated the model with the blocks it is allowed to choose from, a MIP is solved to find the best subset of blocks to mine in the current period. The blocks from this solution are then added to the model as "fixed", their resource usage added to the total resource limit and the whole process starts again by computing and selecting cones for the next period.

As this process, with suitable values chosen for $p$, $\rho$ and $\mu$, is relatively fast; $n$ random solutions are first generated, with the best being selected for further local improvement.

## 4.2 Local improvement heuristic

If there was no discount rate included in the problem description, then any feasible solution to the problem which mines all blocks identified in the solution to the UPIT problem within the given time periods would be optimal. The inclusion of a discount rate says that the later a block is mined, the closer to zero its value becomes; meaning that better solutions will have positively valued blocks mined earlier and negatively valued blocks later. It is this principle behind the local improvement phase of the proposed GRASP algorithm.

Operating similarly to the *bubblesort* algorithm [6], a window of size $\omega$ (typically two periods long) is moved back and forth along a time-expanded version of the randomly constructed solution, searching for high-valued blocks that can be moved earlier in time and loss-making blocks that can be moved to later (Figure 7). Once all possible swaps have been made within a window, that window is then moved along while still overlapping at least one period so that blocks swapped in the previous search may potentially be swapped further in the current one.

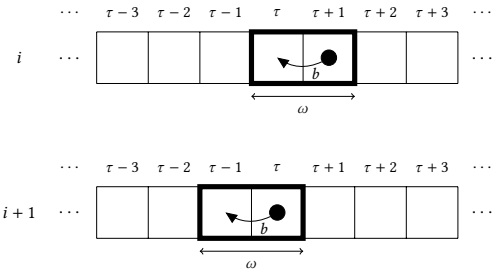


**Figure 7: Window slides along the solution with each iteration; unfreezing $\omega$ periods at a time and allowing block $b$ to move from period $\tau + 1$ to $\tau - 1$ over two iterations.**

Although this method requires a full, time-expanded solution, because only the blocks that are mined within the current window are included in the model; the model is much smaller than that needed to solve the original problem and therefore tractable, even for very large problem sizes. Blocks that are included are free to take any time value within the window (that does not violate any precedence or resource constraints), while blocks that are not included are fixed at their current value in the solution.

The random construction phase aims to use as much of the resource limit as possible for each period meaning that swapping a block from one period to another which is near its resource limit requires a reciprocal swap from a block in the period it is trying to swap to - otherwise the resource constraints will be violated. For this reason, it makes sense to start sliding the window from the last period rather than the first, as the last period will not have reached its resource limit and therefore will allow blocks to be swapped into it without needing to find a reciprocal swap. This, in turn, frees up space in the previous period so that blocks from the period before that can be swapped easily, and so on down the line, making the whole process run faster.

### 4.3 Algorithm

Algorithm 1 details the procedure by which the *minePS+LI* algorithm functions.

$t_{max} \leftarrow$ number of periods in problem
$R_t \leftarrow$ amount of resources allowed in period $t$
initialise MIP model
fix all variables to 0
**for** $t = 0$ to $t_{max}$ **do**
  calculate cone values
  **while** used resources $< \mu \times R_t$ **do**
    add cones to model by unfixing their member blocks
  **end while**
  solve MIP model
  fix blocks in solution to present value
  update resource constraints
**end for**
**while** not converged or time limit reached **do**
  **for** $t = t_{max} - \omega - 1$ down to 0 **do**
    initialise empty MIP model
    add all blocks in periods $t$ to $t + \omega$ to model
    solve model using current solution as warm start
  **end for**
**end while**

**Algorithm 1:** Pseudocode for *minePS+LI* algorithm.

## 5 EXPERIMENTAL DESIGN

This Section gives the details of the datasets from minelib and outlines the experiments that were performed.

### 5.1 Datasets

Table 1 shows the characteristics of the problem instances used to test the algorithms. The first column gives the instance name; the second column gives the number of physical blocks in each orebody model; the third gives the number of block precedences in the orebody model; the fourth displays the number of decision variables requried, based on the the formulation given in Equations (1-6); and finally, the last column gives the total number of constraints in this same formulation. It can be seen from this Table that all problem instances except for `newman1` have too many variables and constraints to be tractable for conventional mathematical solvers.

The full minelib dataset contains 11 different instances, 6 of which have been chosen for comparison in this research. Several instances were omitted due to several factors. No `.pcpsp` problem file was available (`p4hd`), problem instance references more than two resources (`w23`) and problem instances were too large for the algorithm in its current form (`sm2`, `mclaughlin_limit`, `mclaughlin`).

**Table 1: Characteristics of *minelib* datasets.**

| Instance | Blocks | Precedences | Periods | Variables | Constraints |
|---|---|---|---|---|---|
| newman1 | 1,060 | 3,922 | 6 | 19,080 | 46,864 |
| zuck_small | 9,400 | 145,640 | 20 | 564,000 | 3,514,440 |
| kd | 14,153 | 219,778 | 12 | 509,508 | 3,203,468 |
| zuck_medium | 29,277 | 1,271,207 | 15 | 1,317,465 | 20,502,708 |
| marvin | 53,271 | 650,631 | 20 | 3,196,260 | 16,422,004 |
| zuck_large | 96,821 | 1,053,105 | 30 | 8,713,890 | 40,694,384 |

### 5.2 Experiments

The experiments were run on an *Intel® Core™ i5-6200U* processor (2.30GHz) with 8GB RAM running Linux. All code was implemented in C++ with GCC-4.8.0. The boost library implementation of the Boykov-Kolmogorov algorithm was used to solve the UPIT problem and CPLEX Studio 12.7 operating with up to 4 parallel threads was used to solve the MIPs.

Comparisons were made on instances of the *minelib* dataset between a greedy algorithm, the random solution construction algorithm without the local improvement, the full GRASP heuristic with random solution construction and local improvement and the published state-of-the-art results from *minelib*.

The greedy search algorithm is deterministic, and therefore its solution does not change with the number of runs, so it was only run once for each instance. Two versions of the proposed heuristic were tested; *minePS* is the period-by-period solution construction heuristic and *minePS+LI* is the construction phase with additional local improvement heuristic.

The experiments designed to test the random solution construction were run for 30 times each instance with the mean and standard deviation being recorded. The experiments that test the full *minePS+LI* algorithm were run 10 times each, with a stopping condition of either no change to objective value after two full passes of the sliding window, or 10 hours wall time.

Finally, experiments testing the sensitivity of the $\omega$ parameter were run to compare the amount of wall time taken to perform one full pass of the window-based local improvement heuristic. The same starting solution was used for each run and each run was performed 5 times for $\omega = 2$ and $\omega = 3$ for the given instances with the mean and standard deviation reported.

The parameter settings for all of the experiments were $p = 0.5$, $\rho = 0.4$, $\mu = 1.1$, $n = 5$ and $\omega = 2$ (except for the exepriments testing $\omega$). Section 4.1 gives a description of these parameters and Section 6.2 discusses the choice of their values.

## 6 RESULTS AND DISCUSSION

This section gives the results of the experiments outlined in the previous section and discusses the outcomes.

### 6.1 Experiments on *minelib* dataset

Table 2 shows the results of the experiments described in the previous section. As no time information was published for the *minelib* results, comparison had to be on solution quality alone. Due to the randomised nature of the algorithms, the mean and standard deviation was recorded for the *minePS* and *minePS+LI* results. The LP upper bound, reported in minelib and computed using a modified version of the Bienstock & Zuckerberg algorithm [1] is shown to give an indication of the quality of the heuristic results.

Table 2 shows that for 4 out of 6 instances, *minePS+LI* is able to beat the current state of the art published results as obtainable on the *minelib* website [5]. In the two instances that the algorithm was unable to match the published results, the objective value is

**Table 2: Results on *minelib* dataset instances. Mean and standard deviation reported only for *minePS* and *minePS+LI*, as *greedySearch* is deterministic and *minelib* had no other information reported.**

| | LP Upper bound | *minelib* | *greedySearch* | *minePS* | | *minePS+LI* | |
|---|---|---|---|---|---|---|---|
| Instance | | | | mean | std. dev | mean | std. dev. |
| newman1 | 2.45E+07 | 2.37E+07 | 2.16E+07 | 2.38E+07 | 1.83E+05 | **2.41E+07** | 0.0 |
| zuck_small | 9.06E+08 | 8.72E+08 | 6.30E+08 | 7.94E+08 | 3.05E+06 | **8.91E+08** | 1.24E+06 |
| kd | 4.11E+08 | **4.07E+08** | 4.87E+07 | 3.73E+08 | 3.49E+06 | 4.02E+08 | 6.60E+05 |
| zuck_medium | 7.51E+08 | 6.76E+08 | 3.54E+08 | 6.67E+08 | 4.16E+06 | **7.28E+08** | 4.48E+06 |
| marvin | 9.12E+08 | 8.86E+08 | 5.30E+08 | 7.91E+08 | 6.61E+06 | **8.97E+08** | 2.91E+06 |
| zuck_large | 5.79E+07 | **5.73E+07** | 5.50E+07 | 5.12E+07 | 1.20E+05 | 5.70E+07 | 1.22E+05 |

**Table 3: Times (wall, seconds) taken to complete a full pass of the window based local improvement heuristic with $\omega = 2$ and $\omega = 3$. $\Delta_{obj}$ is the percentage of improvement in solution quality over the full pass.**

| | $\omega = 2$ | | | $\omega = 3$ | | |
|---|---|---|---|---|---|---|
| Instance | mean (s) | std. dev. (s) | $\Delta_{obj}$ (%) | mean (s) | std. dev. (s) | $\Delta_{obj}$ (%) |
| newman1 | 1.20 | 0.45 | 0.70 | 2.60 | 0.55 | 0.91 |
| zuck_small | 17.20 | 0.45 | 1.50 | 215.40 | 6.73 | 2.50 |
| kd | 20.75 | 4.92 | 8.22 | 230.00 | 11.98 | 12.56 |
| zuck_medium | 678.20 | 7.43 | 2.53 | 8,466.60 | 103.44 | 6.22 |
| marvin | 32.80 | 2.28 | 1.37 | 137.40 | 3.91 | 2.46 |
| zuck_large | 344.08 | 15.85 | 4.05 | 1342.67 | 44.79 | 6.34 |

**Table 4: Time (wall, seconds) taken to achieve same objective value or better than published *minelib* results using window-based local improvement heuristic.**

| Instance | Time (s) |
|---|---|
| newman1 | < 1.0 |
| zuck_small | 1343 |
| kd | N/A |
| zuck_medium | 523 |
| marvin | 1565 |
| zuck_large | N/A |

within 1 percent of the upper bound provided by the LP relaxation, and therefore very close to optimal. In these cases *minePS+LI* is able to produce results within 2 percent of the LP upper bound.

newman1 reported a standard deviation of 0.0 because the result is optimal, this is provable as the problem size is small enough for CPLEX to solve the full, time-expanded formulation without any decomposition. The rest of the instances report heuristic results which are feasible and within the LP bounds, but are almost certainly sub-optimal. The reason for convergence in these cases is most likely window size; having $\omega = 2$ means that it is hard to swap a block to a time period two (or more) steps away, if it does not give an immediate increase in objective value when swapped to a period one step away. Increasing the window size allows blocks to make bigger jumps between periods which increases the chances that the algorithm is able to escape local optima and allows the algorithm to find higher quality solutions. This comes at a significant cost of computation time, as increasing the window size increases the number of blocks included in the model. This is clearly illustrated in Table 3, where it can be seen that although the solution quality can double with an increase in $\omega$, the time taken to compute the result can multiply by up to 10 times.

Finally, it is worth noting that although the algorithm was allowed up to 10 hours wall time to complete, in the cases where the algorithm was able to beat the minelib results, the time taken to achieve a better objective value was significantly less than the full 10 hours allowed. Table 4 gives a summary of the time needed for the local improvement heuristic to find a solution with the same or better objective value than the reported results on minelib.

## 6.2 Parameter choice

The majority of the parameters are used in the random solution construction phase of the algorithm ($p$, $\rho$, $\mu$ and $n$) whereas only one ($\omega$) is used in the local improvement heuristic. The first 3 ($p$, $\rho$ and $\mu$) are used to control the random cone selection, and have the biggest impact on the quality and diversity of generated solutions.

$p$ controls the probability with which a cone is selected; because the cones are ordered by value, setting this value to very high will mean that cones of high value will usually be selected, however there will be a tendency for the same cones to be selected every time, meaning diversity of solutions will suffer. Setting $p$ to a very low value will result in a great diversity of solutions but the trade off will be that they will not necessarily be very high quality, as low valued cones will have a much greater chance of being selected as high valued ones. A good trade-off between quality and diversity was found to be $p = 0.5$. $p$ has no impact on the speed of the search, other than at very low values it can cause the cone selection process to skip many cones, but this increase is negligible.

The parameter which controls how much of the available resources a cone can consume before it is elligible for selection is $\rho$. If this is set very low the cones will typically be much smaller; allowing more to be selected and increasing the breadth of the search which means that the algorithm has more options of areas to find high valued cones in the following periods. Conversely, setting $\rho$ to a very high value will result in much larger cones with fewer being selected; this results in a much more directed search toward high value blocks that are reachable in the current period, but restricts the space in which the algorithm can select from in subsequent periods. A good trade-off between directed and broad search was found to be $\rho = 0.4$. $\rho$ has no impact on the speed of the search except in the case where $\rho$ is very high and a cone consumes 90%

of the allowed resources, so the algorithm must select two cones, adding enough blocks to consume 1.8 times the allowed resources and so slowing down the computation.

The parameter $\mu$ has the biggest impact on computation time during solution construction; $\mu$ is a multiplier of the allowable resources for that period and effectively controls how many blocks can be added to the model for a period. Setting $\mu = 1.0$ means that cones can only be added until there are enough blocks to satisfy the resource limits for the current period (plus some extras that come from having to add a full cone); this means that the MIP solver will be reasonably quick, because most of the blocks included in the model will be mined as there are not many others to select from once the resource limit is reached. Making $\mu$ take a higher value increases the time taken to solve the MIP because there are more blocks added to the model so the size of the model will be much greater and the solver must search within a space of many more possible combinations. In order to ensure faster computation times and allow for multiple solutions to be generated before improvement a value of $\mu = 1.1$ was used in the experiments.

The final parameter used in the random solution construction phase is $n$ which controls the number of solutions that should be generated before one is selected for improvement. This parameter has a linear effect on the time taken to generate a solution and therefore to solve the problem. A good value was found to be $n = 5$ and this was used in the experiments in this paper.

The only parameter which is used in the local improvement heuristic is the one which controls the window size, $\omega$. This parameter has a big effect on both the solution quality and the convergence time; the value of $\omega$ providing a trade-off between the two. Increasing $\omega$ allows the algorithm to move blocks over larger distances in time, letting it to explore a more diverse set of solutions at once so it is less likely to get trapped in local optima and therefore produces solutions of a higher quality. The price that is paid for this is that increasing window size significantly increases the time taken for each iteration of the window search (Table 3). This happens for two reasons; the first is that more blocks are included in the model; the second is that for each block there are more time periods to choose from, increasing the number of variables in the model.

## 7 CONCLUSIONS AND FUTURE WORK

This paper has presented a GRASP-MIP hybrid algorithm for solving the precedence constrained production scheduling problem in large scale mining applications. The proposed algorithm was developed using a period-by-period decomposition approach with a window-based local improvement heuristic. Comparative experiments were carried out on well-known benchmark problems against the current state-of-the-art, a simple greedy heuristic and two incarnations of the proposed algorithm; one with the local improvement heuristic and one without. In 4 of the 6 instances tested, *minePS+LI* was found to produce better results than the current published state-of-the-art, and within 2% of the LP upper bound for the other two.

While the *minePS+LI* was able to beat the published state-of-the-art results in most of the tested instances, the idea is relatively simple and there is still a lot of scope for improvement to both efficiency of computation and quality of solutions.

Solution quality could be improved by embedding the algorithm within a variable neighbourhood search (VNS) framework by increasing the window size when the algorithm converges prematurely. Once a better solution is found, the window size can be reduced again to allow faster computation.

One possibility to improve computation time with larger window sizes is to employ some of the ideas from the random cone selection used in the solution construction. Presently, the algorithm includes all blocks from every period in the current window, but if that number can be reduced to only include the "important" blocks, it could potentially allow for larger window sizes.

Another avenue to investigate, regarding reducing computation time is to use aggregation techniques like those in [8]. By grouping blocks that will most likely always be mined (or not mined) together, this will reduce the number of variables in the model and speed up the operation of the MIP solver.

Finally, in order to solve the *very* large scale instances in the dataset, some work could be done on re-structuring the code so that it does not initialise the entire model at the start and then freeze and un-freeze variables as it needs; but rather constructs a small part of the full model with every period, only consisting of the blocks that are members of the included cones. This might slow the MIP solver down as it needs to reconstruct a model for every period, but hopefully it will allow the larger instances to be solved.

## REFERENCES


[1] Daniel Bienstock and Mark Zuckerberg. 2010. Solving LP Relaxations of Large-Scale Precedence Constrained Problems. In *Integer Programming and Combinatorial Optimization*, Friedrich Eisenbrand and F. Bruce Shepherd (Eds.). Number 6080 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1–14.

[2] Andreas Bley, Natashia Boland, Christopher Fricke, and Gary Froyland. 2010. A strengthened formulation and cutting planes for the open pit mine production scheduling problem. *Computers & Operations Research* 37, 9 (Sept. 2010), 1641–1647.

[3] Yuri Boykov and Vladimir Kolmogorov. 2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence* 26, 9 (2004), 1124–1137.

[4] Renaud Chicoisne, Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno, and Enrique Rubio. 2012. A new algorithm for the open-pit mine production scheduling problem. *Operations Research* 60, 3 (2012), 517–528.

[5] Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno, and Alexandra N. Newman. 2012. Minelib: A Library of Open Pit Mining Problems. *Annals of Operations Research* 206(1) (2012), 91–114.

[6] Edward H Friend. 1956. Sorting on electronic computer systems. *Journal of the ACM (JACM)* 3, 3 (1956), 134–168.

[7] Dorit S. Hochbaum and Anna Chen. 2000. Performance Analysis and Best Implementations of Old and New Algorithms for the Open-Pit Mining Problem. *Operations Research* 48, 6 (Dec. 2000), 894–914.

[8] Enrique Jélvez, Nelson Morales, Pierre Nancel-Penard, Juan Peypouquet, and Patricio Reyes. 2016. Aggregation heuristic for the open-pit block scheduling problem. *European Journal of Operational Research* 249, 3 (2016), 1169–1177.

[9] Helmut Lerchs and Ingo F. Grossman. 1964. Optimum design of open-pit mines. In *Operations Research*, Vol. 12. Inst. Operations Research Management Sciences.

[10] C. Meagher, R. Dimitrakopoulos, and D. Avis. 2014. Optimized open pit mine design, pushbacks and the gap problem–a review. *Journal of Mining Science* 50, 3 (May 2014), 508–526.

[11] Gonzalo Ignacio Muñoz Martínez. 2012. Modelos de optimización lineal entera y aplicaciones a la minería. (2012).

[12] Jean-Claude Picard. 1976. Maximal closure of a graph and applications to combinatorial problems. *Management science* 22, 11 (1976), 1268–1272.

[13] Mauricio GC Resende and Celso C Ribeiro. 2010. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In *Handbook of metaheuristics*. Springer, 283–319.

[14] Gaurav Singh, David Sier, Andreas T Ernst, Olena Gavriliouk, Rob Oyston, Tracey Giles, and Palitha Welgama. 2012. A mixed integer programming model for long term capacity expansion planning: A case study from The Hunter Valley Coal Chain. *European Journal of Operational Research* 220, 1 (2012), 210–224.