An Improved Merge Search Algorithm For the Constrained Pit Problem in Open-pit Mining

Angus Kenny
School of Science,
RMIT University
angus.kenny@rmit.edu.au

Xiaodong Li
School of Science,
RMIT University
xiaodong.li@rmit.edu.au

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

Yuan Sun
School of Science,
RMIT University
yuan.sun@rmit.edu.au

ABSTRACT

Conventional mixed-integer programming (MIP) solvers can struggle with many large-scale combinatorial problems, as they contain too many variables and constraints. Meta-heuristics can be applied to reduce the size of these problems by removing or aggregating variables or constraints. Merge search algorithms achieve this by generating populations of solutions, either by heuristic construction [4], or by finding neighbours to an initial solution [12]. This paper presents a merge search algorithm that improves the population generation heuristic in [12] and utilises a variable grouping heuristic that exploits the common information across a population to aggregate groups of variables in order to create a reduced sub-problem. The algorithm is tested on some well known benchmarks for a complex problem called the constrained pit (CPIT) problem and it is compared to results produced by a merge search algorithm previously used on the same problem and the results published on the *minelib* [9] website.

CCS CONCEPTS

• **Mathematics of computing → Combinatorial optimization**;
• **Applied computing → Operations research**;

KEYWORDS

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

ACM Reference Format:
Angus Kenny, Xiaodong Li, Andreas T. Ernst, and Yuan Sun. 2019. An Improved Merge Search Algorithm For the Constrained Pit Problem in Open-pit Mining. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321812

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321812

1 INTRODUCTION

Many modern optimisation problems are difficult for conventional mixed-integer programming (MIP) solvers because of the large number of variables and constraints they contain. Real-world problems such as those found in the field of open-pit mining optimisation are particularly susceptible to these problems, due to their massive physical and chronological scale and often complex constraints. Various techniques have been developed, called hybrid meta-heuristics [17], in order to decompose, reduce or otherwise transform these intractable search spaces, by removing redundant variables and constraints, such that they can be solved by conventional solvers. Among this family of techniques are column generation [8], which initially considers only a restricted subset of the variables, with more added as they are needed; Benders' decomposition [1], which splits a large problem into smaller sub-problems that are then solved and recombined; branch-and-cut [5], which uses the ability of meta-heuristics to find feasible solutions quickly in order to prune a branch-and-bound tree; and many more.

One more recent branch of research in this area concerns so-called merge search algorithms [4, 12, 18, 19]. These algorithms use information from a population of solutions to determine a subset of decision variables to be included in a reduced sub-problem that is then solved using a MIP solver. Often, these variables are included in the reduced sub-problem individually, so for large problem instances, it is still impossible to search a very large neighbourhood. This paper presents an algorithm, *ImprovedMerge*, that builds on the *ParallelMerge* algorithm in [12], and gives an improved method of generating the population which encourages the creation of a more compact and effective reduced sub-problem. It also uses a variable grouping heuristic similar to that in [18] that exploits population information to aggregate variables into groups and allows for further reduction in the problem size, without sacrificing neighbourhood size; or, alternatively, allowing for a larger neighbourhood to be searched for the same computational resources. The trade-off for this is a reduction in the resolution of the search and this is discussed in Section 5.

This improved algorithm is applied to solve a real-world problem from the well-known *minelib* dataset [9], called the constrained pit (CPIT) problem. CPIT combines the two most important problems in open-pit mining: what to mine; and when to mine it. The mining industry is a very important industry in Australia and around the

world, and because the problems are on such a large-scale, it is critical to be able to minimise costs wherever possible. Due to their size and structure, open-pit mining problems are well-suited to merge search algorithms.

Experiments are performed that compare *ParallelMerge* with *ImprovedMerge* and against the published results in *minelib*. *ImprovedMerge* is found to produce better quality results than *ParallelMerge* in all instances and is better than the publish *minelib* results in five of the six instances.

2 BACKGROUND

This section gives a description of the CPIT problem, introduces ways in which it can be modelled and gives a brief survey of how other researchers have approached solving it, as well as previous work from other researchers that relates to merge search.

2.1 Related work

This paper extends the *ParallelMerge* algorithm for the CPIT problem [12], which itself is built upon three main works [4, 13, 18]. In their previous work, [13], Kenny et al. presented a greedy randomised adaptive search procedure (GRASP) framework for solving the precedence constrained production scheduling problem (PCPSP) in open-pit mining. The GRASP algorithm greedily constructs solutions in a period-by-period fashion which are then improved by solving a reduced sub-problem on a set number of periods using a "sliding window" heuristic. They used a similar method to the cone-based preprocessing technique from this work for the *ParallelMerge* algorithm, and the algorithm that is presented here uses the same method to generate the initial seed solutions.

In [18], Thiruvady et al. introduces a merge algorithm for the PCPSP and shows that it improves several of the best known linear programming (LP) bounds. The "sliding window" heuristic used in [13] is an extension of this work. Thiruvady et al. also describe a method of aggregating variables to reduce the size of the sub-problem to be solved by comparing the values of the variables across the entire population. Similar techniques are applied to solve a scheduling problem in [19]. This variable aggregation heuristic is adopted as part of the algorithm presented in this paper; however the methods of generating the merge population are vastly different between the two studies and so the behaviour of the merge search itself is very different.

Some parallels can be drawn between the merge algorithm described here and in [13] and [18] and the construct, merge, solve and adapt (CMSA) algorithm by Blum et al. [4]. In CMSA, solutions are generated from scratch and their components added to an initially empty sub-instance C'. This sub-instance is solved using an exact solver and any components in C' that have not been useful in previous iterations can be removed by the use of a so-called ageing mechanism. Because the CPIT problem is so large and complex, it is too costly to generate solutions from scratch, and even then it is hard to guarantee that they will be of any good quality. This is the main area in which the merge search algorithms presented in [12, 18] and this paper differ greatly to CMSA. The population for merge search comprises neighbouring solutions to an initial solution; while CMSA generates the solutions, from which it draws their constituent components, from scratch. There is also no need for an ageing mechanism in merge search to regulate the population, as each iteration produces a new set of solutions.

Open-pit mining problems are as old as mining itself; but they have been formalised as mathematical problems since the 1960s. Despite this, very little exists in the literature on how to solve them using meta-heuristic techniques, perhaps because of their large-scale nature. In [7], a topological sorting heuristic is used to repair the solution to the LP relaxation of the CPIT problem. Once a feasible solution is obtained from the fractional LP solution, it is then improved by a local search algorithm.

Jélevez et al. [11] use a heuristic technique to aggregate the blocks of the CPIT problem into larger groups. This reduces the number of variables and constraints in the problem model and allows it to be solved more efficiently. Solutions to this simplified model are then iteratively disaggregated to obtain better solutions.

Extra inequality constraints are added by Bley et al. [3] in order to strengthen the MIP formulation for various open-pit mining problems. Due to the size of the problem instances, this strengthened formulation was not tested on the *minelib* instances, but it is shown that the extra constraints effectively reduce the computation time needed by CPLEX to solve their custom problem instances. For a good survey of research in these related areas, see [10, 15].

In their masters' thesis, Muñoz uses a modified version of the Bienstock and Zuckerberg algorithm [2] to solve the LP relaxation before using a topological sorting algorithm to produce a feasible solution, similar in method to Chicoisine et al. [7]. It is the results from these experiments that are published on the *minelib* website, and used for comparison in this paper.

2.2 Problem description

This work extends [12], which itself builds on [13], which uses a GRASP algorithm to solve the PCPSP. This section will give a brief introduction to the CPIT problem to which the algorithms in this paper are applied to solve. The reader is directed to [13] for a detailed description of the properties and constraints involved in general open pit mine planning problems, and [12] for a detailed description of the CPIT problem and how it relates to the PCPSP.

The CPIT problem is a simplified abstraction of two of the most significant tasks in open-pit mining: deciding what to mine, and deciding when to mine it. It divides the orebody of the mine into blocks, each given a value based on its ore content, and relates each of the blocks to its neighbours by way of precedence constraints.

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.
(2) The predecessor blocks must be mined in the same period or earlier.
(3) The resource limits consumed in mining blocks must not be violated.

Although this problem is characterised as a "real-world" problem, it is worth noting that the CPIT problem is a significantly simplified abstraction of the day-to-day considerations and constraints involved in open-pit mining. There are many more factors that need to be taken into account if this problem is to be considered a true model of reality such as uncertainty in the ore composition of

blocks, position and cost of moving machinery, complex pit shapes and other operational and business constraints.

2.3 Mathematical formulation of CPIT

The CPIT problem consists of the following sets and parameters:

- B the set of blocks;
- T the set of time periods;
- 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_{bt} the profit for mining block b at time t (can be negative if it is a cost only). For the minelib data this is simply $\frac{p_b}{(1+\text{discount})^t}$ for some base cost p_b and a discount value typically in $(0, 1)$;
- q_{br} the amount of resource r required by block b; 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)

Using this notation, the problem is written as:

$$\max \quad \sum_{b \in B} \sum_{t \in t} p_{bt}(x_{bt} - x_{bt-1}), \tag{1}$$

s.t.,

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

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

$$\sum_{b \in B} q_{br}(x_{bt} - x_{bt-1}) \leq \overline{R}_{rt} \qquad \forall\, r \in R, t \in T, \tag{4}$$

$$x_{bt} \in \{0, 1\} \qquad \forall\, b \in B, t \in T. \tag{5}$$

Note: for correctness, in (4) for the first time period, no previous x is to be subtracted. See [12] for a full discussion of this formulation.

2.4 Merge search

The algorithm in this paper builds on the *ParallelMerge* algorithm that is detailed in [12], so the reader is directed there for a full description of the algorithm; however, a brief outline will be given here along with the key findings of that paper.

ParallelMerge achieves a reduction in the problem it is applied to by fixing the bulk of its decision variables and focusing only on a smaller subset of variables. This subset is determined by first heuristically constructing an initial solution of reasonable quality, and then using a local search algorithm to generate a population of neighbouring solutions. This population of solutions are then "overlaid" on each other to determine all of the decision variables that are different between the initial solution and each of the solutions in the population. This set of variables is the set that the reduced sub-problem focuses on, and a MIP solver will determine the best solution available in this neighbourhood.

The basic merge search algorithm starts off with a single initial seed solution and goes through a process of generating a population, merging and finally solving the reduced sub-problem using a MIP solver. The solution to this reduced sub-problem becomes the initial seed solution to the next generation. The parallel version is similar to this, except it starts with n initial seed solutions and runs through the cycle of generate, merge, solve with each of these solutions in parallel, before finally merging all of the populations and performing one final solve.

2.5 Solving CPIT with merge search

Because the structure of the CPIT problem necessitates there being many variables that will always be 0, it was found in [12] that the CPIT problem was a perfect candidate to apply a merge search algorithm to. When the problem is represented as a graph (Figure 1), a solution to the problem becomes a closure on that graph and a partition of the variables that are 0 and the variables that are 1.

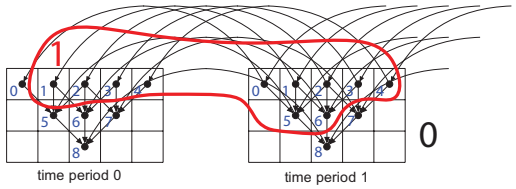


Figure 1: A solution to the CPIT problem as a closure. Here blocks 1, 2, 3 and 4 are mined in period 0; and blocks 0, 5 and 6 are mined in the second period.

The initial solutions were constructed using a similar method to that used in their previous work on the PCPSP [13], where cones of blocks are computed and ranked according to their value and resource usage and then mined heuristically until the resource limits were reached for each period, and then the cones are recomputed and ranked for the next period. This process continues until all the blocks are mined or there are no more periods left to mine.

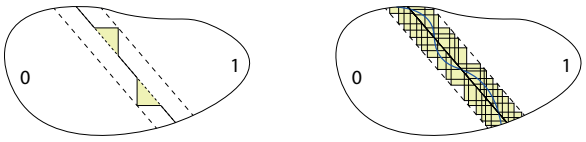
A simulated annealing [14] based algorithm was used as a local search technique to generate the population of neighbouring solutions. This algorithm is based on an abitrary block swap operator, where a block is chosen at random and its cone is computed and all blocks in that cone are swapped to the nearest period. Sometimes this puts the resource limits out of balance, so another swap is performed to try and balance it out, and this continues until the solution is feasible or a predetermined number of swaps is reached — at which point the move is abandoned and a new one attempted.

It was found during these experiments that the simulated annealing heuristic was unnecessary and that purely random swaps would suffice, so the simulated annealing heuristic was not used for this paper, however elements of it (such as the arbitrary block swaps) were kept.

2.6 Variable grouping heuristic

Thiruvady et al. [18] give a variable grouping heuristic for use with merge search algorithms. It uses information from across the entire population to determine variables that can be grouped together and treated as a single variable, in order to reduce the size

of sub-problem. Figure 2a gives an abstraction of the set of decision variables for a binary optimisation problem. A solution to any such problem can be represented as a partition between the variables that take the value 0 and those that take the value 1.



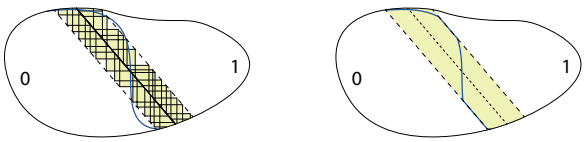
(a) Neighbouring solution with differing subsets **(b) Neighbourhood region with optimal solution**

Figure 2: Each neighbouring solution contains a differing subset of variables to its initial solution. Merging a population of solutions allows a picture of the neighbourhood region to emerge.

In Figure 2, the initial solution is shown as the short-dashed line in the middle. The neighbourhood of this initial solution is defined by the region that contains all possible solutions that are reachable in a given number of local moves and is represented by the longer-dashed lines. A neighbouring solution is defined as one that can be produced by performing a series of local moves on the initial solution; it is represented by the solid black line. The shaded regions are the decision variables that differ in value between the initial solution and the neighbouring solution - referred to here as the differing subsets.

By sampling and merging a sufficiently large and diverse population from the possible neighbouring solutions, a picture of the neighbourhood region can be built up by finding the union of all of the differing subsets. A reduced MIP sub-problem can then be created by fixing all variables in the 0 region to 0, all variables in the 1 region to 1 and only allowing the solver to focus on those variables in the differing subsets. When solved, provided it lies within the neighbourhood region, the reduced sub-problem is able to find the optimal solution (Figure 2b).

The problem then lies when the optimal solution is outside the neighbourhood boundary. This can happen when the initial solution is of poor quality, or the neighbourhood being used is too small (Figure 3a).



(a) Optimal solution is outside neighbourhood **(b) Reduced sub-problem produces sub-optimal solution**

Figure 3: When the optimal solution is outside of the neighbourhood, the reduced sub-problem cannot find it; even with perfect neighbourhood information.

Figure 3b illustrates that even as the size of the population tends towards infinity, although the neighbourhood boundary is approximated better, the reduced subproblem is unable to be solved to optimality. The simple solution to this problem is to increase the size of the boundary region by either increasing the number of moves, or the size of each move - whichever is appropriate to the local search algorithm being used - however, this requires more free variables in the reduced sub-problem and hence more computational resources to solve it.

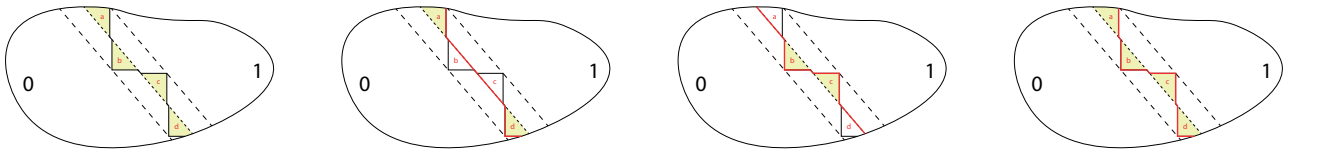
3 IMPROVED MERGE SEARCH FOR CPIT

This section gives a description of *ImprovedMerge*, the merge search algorithm presented in this paper that builds on *ParallelMerge* from [12]. It describes the improvements made to the population generation heuristic that create a more compact and effective reduced sub-problem, and also the variable grouping heuristic that helps to reduce the sub-problem further.

3.1 Population generation

The method of population generation employed by *ImproveMerge* is more greedy than that in *ParallelMerge* and differs in a number of ways. *ParallelMerge* uses a swap operator controlled by a simulated annealing algorithm [14] that accepts or rejects arbitrary swaps according to a probability function and temperature variable. The temperature variable works by increasing the probability that a "bad" move will be accepted early on in the search, reducing this probability as the search goes on and the temperature "cools". These heuristics are generally useful in avoiding local optima, as they will still accept moves that improve a solution, but are able to explore more of the search space by accepting moves that temporarily worsen it. In the case of the CPIT problem however, the number of possible moves from any problem state is so large and the size of the moves so small, that any advantage gained by accepting a worsening move is outweighed by the cost of including those extra variables in the reduced sub-problem. To combat this, the improved algorithm presented here includes a number of changes to the local search heuristic that encourages adding variables to the reduced sub-problem that will lead to better quality solutions.

ImprovedMerge does away with the simulated annealing altogether and uses a (semi-guided) random walk. As each individual move is very computationally cheap, a set of potential moves is generated at the start of each iteration; the best move (i.e., the one that has the best ratio of impact on objective value to impact on resource consumption) being chosen from this set and used as the initial move. This initial move is likely to have caused the resource constraints to be violated in one of the periods; so arbitrary swap moves are performed around the period boundary in order to repair the solution. This repaired solution is added to the population if and only if the set of moves contains at least one move that would improve the objective value of the solution in its own right, regardless of whether the over-all objective value is better than the seed solution; otherwise, it is discarded and nothing added to the population for that iteration. The reason for this is that even though the final solution may be of worse quality, it still contains some variables that when included in the reduced sub-problem could produce an improvement. Furthermore, if the generated solution is of better quality than the seed solution, it replaces the seed solution in the population so that the next solution to be generated will



(a) Solutions $\{a, b\}$ and $\{c, d\}$ **(b) Solution with groups $\{a, d\}$** **(c) Solution with groups $\{b, c\}$** **(d) Solution with groups $\{a, b, c, d\}$**

Figure 4: By selecting different combinations of variable groups in the reduced sub-problem, new solutions can be generated.

start from a better starting point. In *ParallelMerge* the same seed is used to generate the entire population.

The effect of these changes means that fewer variables are added to the reduced sub-problem, and those that are added are more likely to produce better solutions when solving the sub-problem. By greedily updating the starting seed solution, it means that any previously accepted moves are included initially, and removes the likelihood of repeating the same moves while also allowing future moves to be built on top of previous ones. Without the greedy seed updating, all solutions in the population remain within one step of the initial solution.

3.2 Variable grouping

In order to allow for the largest neighbourhood possible in the reduced sub-problem with the same amount of computation required to solve it, decision variables can be grouped together and consdered as a single variable in the reduced sub-problem.

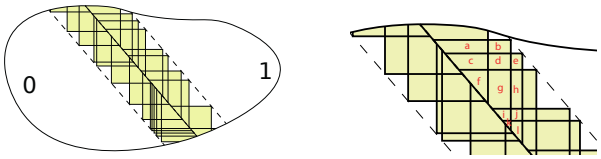
There are certain decision variables that when included in a solution, imply that ones around them should also be included, and so it makes sense to group them together as a single variable. In the case of the CPIT problem, this can be because they are near each other in physical space, or because of precedence constraints; but there are other ways that this can also happen. The structure and nature of each problem will determine the appropriate way for variables to be grouped together.

It was shown previously that generating a new solution that is in the neighbourhood yields a subset of the decision variables that have been changed from their value in the initial solution in order to create this new solution. This information in the differing subsets can be exploited in order to aggregate the decision variables automatically as, in order to generate the new solution, a series of neighbouring moves must be performed on the initial solution, creating a chain of feasible solutions that lead from the initial solution to the solution in question.

For example, if the population consists of two solutions that are considered independent (i.e., there is is no intersection between their differing subsets and no dependency between them in the constraints), all of the variables in each of the differing subsets can be considered as a single group; and as such treated as a single variable in the restricted sub-problem. Figure 4 shows one such example, where one solution is made up of subsets $\{a, b\}$ and the other solution is made up of subsets $\{c, d\}$. This figure shows that along with the initial solution and two population solutions, three more feasible solutions can also be found (in this case, solutions with groups $\{b, d\}$ would not be feasible as it would become unbalanced).

This is obviously a very coarse way of searching for new solutions, and it is hard to guarantee that all of the solutions in the population will be independent from each other, and so allow for each differing subset to be considered as a single group. However, it is possible to use the overlapping nature of these differing subsets to increase the granularity of the groups and so increase the number of possible solutions, some care just needs to be taken in order to preserve precedence (or any other type of) constraints.

Figure 5a shows how the intersections between the differing subsets of variables obtained from each solution in the population can be used to create groups of decision variables that can be considered as a single variable in the reduced sub-problem.



(a) Population of neighbouring solutions **(b) Precedences exist between groups (labelled a-l)**

Figure 5: Overlapping differing subsets in the population creates natural groups of variables, but there still exists the same precedence constraints between groups.

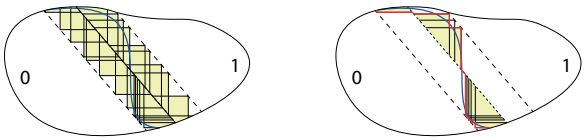
Each yellow region in this figure that is bounded by a thick black line can be considered a group in its own right, and new solutions can be found by the MIP solver by creating various combinations of these groups, paying attention to any constraints that exist in the original problem. Figure 5b shows a subset of the groups in order to illustrate this concept.

Group a cannot be included in a solution to the reduced sub-problem on its own, as there are a number of variables that must precede the variables in this group. So, in order to include the variables in group a into this solution, the groups $\{c, f\}$ must also be included. Similarly, in order to include the groups $\{a, h\}$ into the solution to the sub-problem, the groups $\{c, f, g, i, j, k, l\}$ must also be included in order to satisfy all precedence constraints. This does not take into account any other type of constraint, and any others must be included in the MIP for the reduced sub-problem explicitly.

3.3 Approximating solutions

Given a problem, an initial solution and a population of neighbouring solutions, Figure 6 shows how this grouping heuristic can be used in order to find an approximate solution to an optimal solution, if it exists within the neighbourhood of the initial solution.

A population of solutions is generated using a local search technique and the decision variables are grouped based on the differing



(a) Optimal solution within neighbourhood **(b) Selected group boundaries form new partition**

Figure 6: The boundaries of the included groups from the reduced sub-problem forms a new partition that approximates the optimal solution.

subsets of all solutions in the population; and their intersections. A restricted sub-problem is created with each of the aggregate groups being represented as a single variable in the sub-problem; with all precedences encoded in the MIP model.

The decision variables are aggregated into groups based on the differing subsets from all of the solutions in the population; and their intersections. The reduced sub-problem is solved to determine which groups are included in the new solution, and so which decision variables to include. A mapping between groups and decision variables is kept in a data structure that can be referenced in order to convert the solution to the restricted problem to a solution to the original problem. Groups are included in the solution such that their collective boundary, along with the initial solution, form a new, feasible, partition that approximates the optimal solution.

It can be seen in Figure 6 that the more groups that are in the reduced sub-problem, the finer grained the approximation that can be obtained; however this requires more variables in the MIP and so more computational resources in order to solve it. A balance between the size of these groups and their number is the one of the most important considerations when setting up the reduced sub-problem to be solved. To this end, it could be useful to find some method of further aggregating the groups such that their size can be determined through the use of a parameter to control the granularity of the search.

3.4 Pseudocode

Algorithm 1 gives the pseudocode for *ImprovedMerge*. The constants n_m and n_{pop} are the number of merges and the population size respectively. The function $localSearch(S)$ returns a solution that is within the neighbourhood of S; the function $improving(S)$ returns true if S contains a move that improves the objective value in its own right; the function $merge(\mathcal{P})$ merges population $\mathcal{P}$ and returns the solution to the reduced sub-problem; and, the function $max(S, S')$ returns the solution from the set $\{S, S'\}$ with the highest objective value.

4 EXPERIMENTAL DESIGN

This section gives the details of the datasets from *minelib* [9] and outlines the experiments that were performed.

4.1 Datasets

Table 1 details the properties of the problem instances used to test the algorithm in this paper. The instance name is given in the first column; followed by the number of blocks in the orebody model;

Input: S (Initial seed solution)
for $m = 0$ *to* n_m **do**
  $\mathcal{P} \leftarrow \emptyset$
  for $p = 0$ *to* n_{pop} **do**
    $S' \leftarrow localSearch(S)$
    if $improving(S')$ **then**
      $\mathcal{P} \leftarrow \mathcal{P} \cup S'$
    end
    $S \leftarrow max(S, S')$
  end
  $S \leftarrow merge(\mathcal{P})$
end

Algorithm 1: *ImprovedMerge*

the number of precedence arcs for each instance is provided in the third column; the fourth column gives the total number of time periods available; the number of decision variables is shown in the second last column; with the last column giving the total number of constraints in the problem model.

Table 1: Characteristics of *minelib* [9] datasets.

Instance	Blocks	Precedences	Periods	Variables	Constraints
newman1	1,060	3,922	6	6,360	29,904
zuck_small	9,400	145,640	20	188,000	3,100,840
kd	14,153	219,778	12	169,836	2,807,196
zuck_medium	29,277	1,271,207	15	439,155	19,507,290
marvin	53,271	650,631	20	1,065,420	14,078,080
zuck_large	96,821	1,053,105	30	2,904,630	34,497,840

A few instances from the full *minelib* dataset were omitted due to missing files, referencing more than two resources or being too large for the algorithm in its current incarnation.

4.2 Parameters

By doing away with the simulated annealing aspects of the local search, the number of parameters was able to be reduced significantly. The main parameters used by the improved merge search algorithm are: n_{pop}, the size of the population of neighbouring solutions to be generated; n_i, the size of the set of moves generated to select the initial move from; and n_m, the number of merges performed each run. The values of these parameters were chosen in order to approximate the computational budget of *ParallelMerge* [12].

ParallelMerge uses a n_{pop} value of 20. Because the simulated annealing heuristic in *ParallelMerge* takes longer than the random walk to generate a solution and it also contains more variables in its differing subset, a value for n_{pop} of 1000 was chosen for *ImprovedMerge* as it gives a better picture of the neighbourhood and still can be generated faster than the much smaller population in *ParallelMerge*. There is no equivalent parameter in *ParallelMerge* for n_i, so a value of 2000 was chosen after testing several different candidates. As the simulated annealing heuristic is much slower than the random walk and the reduced sub-problem contained many more variables, Kenny et al. were not able to perform many merge operations in the given computational budget. Due to the speed of the population generation and the problem reduction

Table 2: Results on *minelib* dataset instances. Given are the linear programming (LP) upper bound, the published *minelib* results, the mean objective value (dollars) for each of the algorithms and the standard deviation; and the mean time (seconds) to beat *minelib* or complete a run, and the standard deviation. Only the objective value for *minelib* [9] is given, as this is the only information available.

Instance	LP UB	*minelib*	*ParallelMerge*				*ImprovedMerge*			
			objective	std. dev.	time	std. dev.	objective	std. dev.	time	std. dev.
newman1	2.449E+07	2.348E+07	2.413E+07	1.506E+04	**0.48**	0.13	**2.418E+07**	1.165E+02	1.71	0.32
zuck_small	8.542E+08	7.887E+08	8.390E+08	9.056E+05	25.73	1.76	**8.433E+08**	2.078E+06	**1.67**	0.01
kd	4.095E+08	3.969E+08	4.007E+08	3.339E+06	1,494.92	63.25	**4.088E+08**	1.881E+05	**93.35**	8.55
zuck_medium	7.106E+08	6.154E+08	6.473E+08	1.828E+06	801.25	5.02	**6.584E+08**	8.462E+05	**10.03**	2.54
marvin	8.639E+08	8.207E+08	8.500E+08	9.534E+05	425.49	4.43	**8.539E+08**	2.575E+05	**14.05**	3.63
zuck_large	5.739E+07	**5.678E+07**	5.182E+07	3.196E+05	16,248.44	676.08	5.441E+07	2.558E+05	6368.21	319.57

acheived by the variable grouping heuristic, many more merge operations were possible. *ParallelMerge* used a n_m value of 5, whereas *ImprovedMerge* was able to perform 20 merge operations and still be significantly faster.

4.3 Experiments

The experiments were run on an *Intel® Core™ i5-2320* processor (3.0GHz) with 24GB RAM running Linux. All code was implemented in C++ with GCC-4.8.0, using OpenMP [6] for multithreading. 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.

Using the results published on the *minelib* [9] website as a baseline, comparisons were made between the *ParallelMerge*results published in [12] and *ImprovedMerge*. Each algorithm was run 20 times on each instance, recording the mean objective value and standard deviation of the best produced solution recorded in each run. The average time taken to beat the objective value from *minelib* was also recorded for each instance or, if the algorithm failed to beat *minelib*, the total run-time for the algorithm. The *minelib* data only provides a single objective value, so no mean objective, standard deviation or time data was available for comparison.

5 RESULTS AND DISCUSSION

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

5.1 Experiments on *minelib* dataset

The results of the experiments performed on the *minelib* dataset are presented in Table 2. The linear programming (LP) upper bound, current best *minelib* results [9] and results from Kenny et al. [12] are given for reference and compared against the results of tests on the *ImprovedMerge* algorithm.

Comparing the mean objective values, it can be seen that *ImprovedMerge* produced better quality results than *ParallelMerge* in every instance; the mean of *ImprovedMerge* beating *ParallelMerge* by over one standard deviation in all cases. It also beat *minelib* in all but the largest instance, and for that it closed the gap to the LP upper bound significantly compared with *ParallelMerge*.

The mean times recorded show that *ImprovedMerge* is able to achieve better quality results than five of the six best results

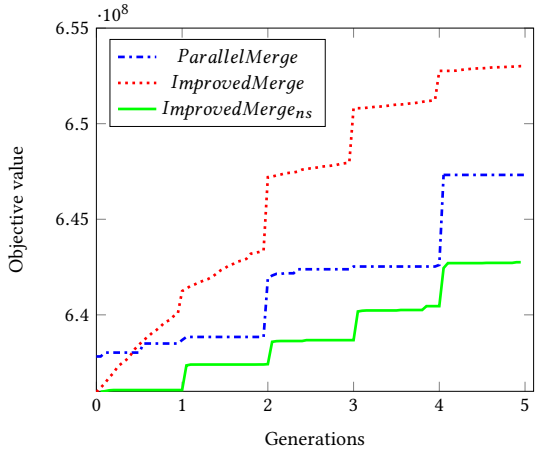


Figure 7: Plot of the current best objective value of *ParallelMerge*, *ImprovedMerge* and *ImprovedMerge*$_{ns}$ (without greedy seed update) on `zuck_medium` for first 5 generations.

from *minelib*, faster in all instances except `newman1`. This is because the simulated annealing heuristic in *ParallelMerge* is able to find a better solution without having to merge at all; whereas *ImprovedMerge* has to perform one merge operation to beat *minelib*.

Figure 7 shows a plot of the current best objective value for both algorithms on `zuck_medium` for the first five generations. The effect of the improved population generation heuristic can be seen clearly in this plot, as the current best objective value for *ImprovedMerge* increases quite rapidly during the population generation period such that even though the search started from a worse quality seed solution, by the time it has reached the second merge operation, it is of better quality than the best solution for *ParallelMerge* up until all of the merge branches converge to produce the final solution. After this point there is a less dramatic increase in seed quality as the local search has found all of the "low hanging fruit" and the merge operation is needed in order to make any significant improvement to the quality of the solutions produced.

The most significant contributor to this improvement appears to be the greedy updating of the initial seed solution, as can be seen by the plot of *ImprovedMerge* against *ImprovedMerge*$_{ns}$. Without updating the seed solution, future moves cannot build upon

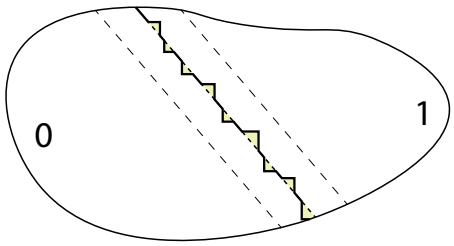


Figure 8: In reality, solutions are much more scattered collections of variables.

previous moves. This means that in a population of, say, ten solutions, the best that the merge operation can ever find is a solution one step away from the initial solution in ten different directions; whereas, if the seed is able to be updated while the population is being generated, it allows the merge operation to find a solution that is ten steps away from the solution in the one direction, if that is the best way to go.

The larger instances of `zuck_medium` and `zuck_large` results still show a large optimality gap. A reason for this could be that because there are so many possible moves from any given state, and each of the moves are so small relative to the over-all mine, that it is causing the groups to be too scattered and not enough of the neighbourhood is being covered by the groups. Figure 2a shows a neighbouring solution where all variables in the differing subset are related to each other and so it is able to reach the absolute limit of the possible neighbourhood boundary. Due to the nature of the initial solution construction heuristic, and the random swaps of the local search, the blocks are not mined in very large chunks; so when swapping between periods, you end up with much smaller 'triangles' of variables (Figure 8). Summed together the areas of these triangles will equal the area of one of the big triangles in Figure 2a, however when merged, they cover a much smaller area of the entire neighbourhood.

5.2 Problem reduction

Table 3 gives the difference in size between the original problem instance and the reduced sub-problem produced by the merge operation, for *ParallelMerge* and *ImprovedMerge*. The table gives the number of variables in the original problem, the number of variables in each of the respective reduced sub-problems, the number of variables covered by the groups in the reduced sub-problem for *ImprovedMerge* and the amount of variables covered as a percentage of the original value.

This table shows that the size of the reduced sub-problem is orders of magnitude smaller for *ImprovedMerge* than it is for *ParallelMerge*. It also shows that *ImprovedMerge* is able to find better quality solutions despite the fact that the number of variables effectively being covered is smaller than *ParallelMerge*. This is likely because of the improvements made to the local search heuristic, where it is being more discriminate about which variables are included in the reduced sub-problem; so, even though there are fewer variables being covered, those variables are more likely to have an improving effect on the solution to the reduced sub-problem.

Table 3: Difference in problem size between the original and the reduced sub-problem produced by the merge operation.

Instance	Original	*ParallelMerge*		*ImprovedMerge*		
		Reduced	%	Reduced	Covered	%
`newman1`	6.36E+03	1.16E+03	18.2	5.98E+02	1.27E+03	19.9
`zuck_small`	1.88E+05	1.50E+04	8.0	7.55E+02	1.98E+03	1.1
`kd`	1.70E+05	1.42E+04	8.4	7.07E+02	3.37E+03	2.0
`zuck_medium`	4.49E+05	3.99E+04	9.1	1.74E+03	9.41E+03	2.1
`marvin`	1.07E+06	1.38E+04	1.3	6.52E+02	1.90E+03	0.2
`zuck_large`	2.90E+06	1.88E+05	6.5	5.31E+02	1.65E+03	0.06

6 CONCLUSIONS AND FUTURE WORK

Open-pit mining problems contain many variables and constraints, making them difficult to solve with conventional MIP solvers. Meta-heuristics are adept at decomposing large problems and obtaining feasible solutions in a reasonable amount of time; however the quality of these solutions is not guaranteed. Hybrid meta-heuristics attempt to mitigate these factors by harnessing the strengths of both types of algorithms.

This paper presented one such hybrid meta-heuristic called *ImprovedMerge*; a merge search algorithm that extends the work in [12] but with improved population generation and variable aggregation heuristics. Unlike the work in [12], *ImprovedMerge* ensures the population comprises only solutions that are likely to improve the over-all quality of the search. This is done by updating its initial seed solution as, and when, it finds a better quality candidate solution; and discarding the candidate otherwise. The variable aggregation heuristic helped to reduce the size of the sub-problem even further by grouping together sets of variables that agreed across the whole population, and treating them as a single variable.

ImprovedMerge was compared against *ParallelMerge* on the well-known *minelib* dataset. In all instances it was shown to produce better quality solutions, in less time than *ParallelMerge* and it was also shown to beat the published *minelib* results in all but one of the 6 instances. There is still a significant gap between the objective value found and the LP upper bound and investigations into the amount the problem is reduced by the algorithm suggests that more of the search space could be considered by increasing the size of the neighbourhood used to generate the reduced sub-problem.

Future work in this direction could focus on finding a method of further aggregating groups or by having a mechanism to control the size of the groups that are included in the reduced sub-problem. Larger groups would allow for a larger neighbourhood to be explored without significantly slowing the search down by having to include more variables. Once the problem has been solved with larger groups, the groups at the boundaries could be further disaggregated to allow for a finer granularity in the search.

ACKNOWLEDGEMENT

This work was partially supported by an ARC Discovery grant (DP180101170) from the Australian Research Council.

REFERENCES

[1] Jacques F Benders. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik* 4, 1 (1962), 238–252.

[2] Daniel Bienstock and Mark Zuckerberg. 2010. Solving LP pelaxations 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.

[3] 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.

[4] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez, and José A. Lozano. 2016. Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization. *Computers & Operations Research* 68 (2016), 75 – 88.

[5] Marco Caserta and Stefan Voß. 2009. Metaheuristics: intelligent problem solving. In *Matheuristics*. Springer, 1–38.

[6] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. 2008. *Using OpenMP: portable shared memory parallel programming*. Vol. 10. MIT press.

[7] 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.

[8] Jacques Desrosiers and Marco E. Lübbecke. 2005. *A Primer in Column Generation*. Springer US, Boston, MA, 1–32.

[9] 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. http://mansci-web.uai.cl/minelib/

[10] 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.

[11] 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.

[12] Angus Kenny, Xiaodong Li, and Andreas T. Ernst. 2018. A merge search algorithm and its application to the constrained pit problem in mining. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. ACM, 8.

[13] 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 the Genetic and Evolutionary Computation Conference (GECCO '17)*. ACM, 1137–1144.

[14] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science* 220, 4598 (1983), 671–680.

[15] 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.

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

[17] E-G Talbi. 2002. A taxonomy of hybrid metaheuristics. *Journal of heuristics* 8, 5 (2002), 541–564.

[18] Dhananjay Thiruvady, Davaatseren Baatar, Andreas T. Ernst, Angus Kenny, Mohan Krishnamoorthy, and Gaurav Singh. 2018. Mixed integer programming based merge search for open-pit block scheduling. *Computers & Operations Research* (under review) (2018).

[19] Dhananjay Thiruvady, Christian Blum, and Andreas T Ernst. 2019. Maximising the net present value of project schedules using CMSA and parallel ACO. In *International Workshop on Hybrid Metaheuristics*. Springer, 16–30.