Comparison of Trajectory and Population-based Algorithms for Optimizing Constrained Open-pit Mining Problem

Iman Rahimi
Faculty of Engineering & IT, University of Technology Sydney, Sydney, Australia
iman83@gmail.com

Theodore Picard
Faculty of Engineering & IT, University of Technology Sydney, Sydney, Australia
theodore.picard@student.uts.edu.au

Andrew Morabito
Faculty of Engineering & IT University of Technology Sydney, Sydney, Australia
andrew.d.morabito@student.uts.edu.au

Kiriakos Pampalis
Faculty of Engineering & IT, University of Technology Sydney, Sydney, Australia
kiriakos.pampalis@student.uts.edu.au

Aiden Abignano
Faculty of Engineering & IT University of Technology Sydney, Sydney, Australia
aiden.j.abignano@student.uts.edu.au

Amir H. Gandomi
Faculty of Engineering & IT, University of Technology Sydney, Sydney, Australia
Gandomi@uts.edu.au

***Abstract*- The problem of open-pit mining optimization is a complex task, often containing many variables. In this paper, we apply a trajectory-based algorithm known as simulated annealing together with a well-known population-based algorithm, genetic algorithm, used to generate solutions for a formulation of the constrained pit problem (CPIT). Three datasets were used to test this simulation, Newman1, zuck_small, and KD. The results show that simulated annealing as a trajectory algorithm possesses a slightly better performance in comparison with the genetic algorithm in terms of profit value.**

Keywords- Metaheuristics, Open-Pit Mining, Genetic Algorithm, Simulated Annealing.

I. INTRODUCTION

An optimization problem, particularly when dealing with mixed-integer programming (MIP), becomes computationally expensive for conventional algorithms to solve. This happens especially when the number of variables and constraints increase, and therefore metaheuristics are proposed to tackle these problems [1] [2][3]. Open-pit mining is one of these fields of optimization problems, where computers show difficulties in solving the complex constraints of the mixed-integer problem in a reasonable amount of time [4] [5]. Open-pit mining is one of the most commonly used surface mining methods to extract minerals from the Earth [6]. This method is nowadays a multi-billion dollar industry [7] [8], and because of this, there has been a huge focus on finding feasible solutions for optimal resource extraction scheduling.

Additionally, metaheuristic methods have been developed and implemented with the goal of reducing the total runtime of optimization problems [9] [10], yet ending with results that are almost as accurate and precise as conventional solving methods. Metaheuristics can be classified into two main categories; Trajectory-based and Population-based methods (Figure 1). A Trajectory-based algorithm uses a single solution that moves toward a design space. In contrast, a Population-based algorithm uses multiple solutions to search for the design space to find an optimal solution. Simulated Annealing (SA) is known as one of the most well-known Trajectory-based algorithms, while the Genetic Algorithm (GA) is recognized as one of the most famous Population-based algorithms.

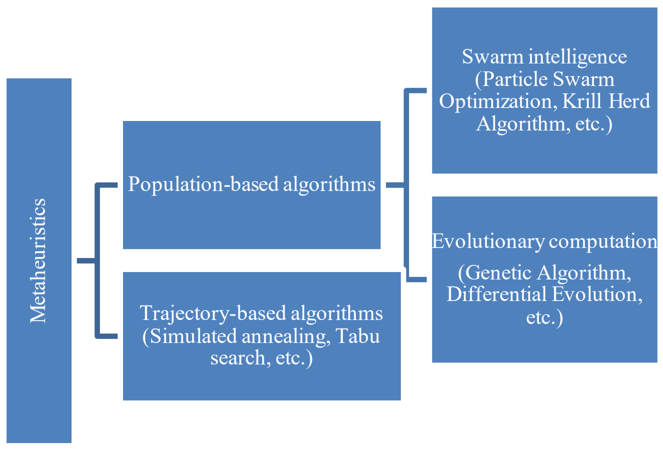
Metaheuristics
Population-based algorithms
Trajectory-based algorithms (Simulated annealing, Tabu search, etc.)
Swarm intelligence (Particle Swarm Optimization, Krill Herd Algorithm, etc.)
Evolutionary computation (Genetic Algorithm, Differential Evolution, etc.)

Figure 1. Metaheuristics classification.

The constrained pit limit problem (CPIT) is an optimization problem formalized by Espinoza et al. [11] regarding open-pit mining. This problem can be simplified by using an analogy of a real open-pit mine, where the goal is to maximize profit. It can be envisioned as a grid of blocks, each with its own numeric value representing the profit from mining the block. The problem has a resource constraint related to the number of blocks that can be excavated during each turn. Furthermore, like a real open-pit mine, blocks must first be mined from the top, before the player can reach the blocks below. Regarding the value of all the blocks, these are known at any point in time. In this study, a SA and a GA are applied to generate solutions to CPIT, where the results are then compared with each other in terms of profit values, using three different datasets.

The rest of the paper is structured as follows: Section II presents the literature review that describes previous works regarding CPIT; Section III describes the methodology and implementation of the performed experiments; Section IV contains the obtained results, and section V presents the main conclusions of the work.

II. LITERATURE REVIEW

In the past, several approaches have been used to optimize the open-pit mine problem. Bienstock and Zuckerberg [12] suggested a method that optimizes a more general form of CPIT, known as the Precedence Constrained Production Scheduling Problem (PCPSP). The algorithm involves a Lagrangian relaxation method while also using specific information regarding the structure of PCPSP, to improve how quickly the algorithm converges on an optimal solution. This method was able to produce quick results when compared to directly using a program to solve the original linear programming problem. However, since Lagrangian relaxation allows the solver to break some constraints, it is not guaranteed that the process produces a feasible solution to the original problem. A version of this algorithm was used to produce some of the example solutions in the dataset used in this paper. Unlike the present paper's experiments, this algorithm uses a program to find an exact solution to the problem rather than the approximate maximum solutions produced with metaheuristic methods [13][14]. Kenny et al. [13] proposed a merge search method for the problem in 2018, having improved it later in 2019 [14].

The merge algorithm is a meta-metaheuristic algorithm that relies on the result of multiple metaheuristics applied to smaller problems. This is achieved using problem reduction techniques, which remove redundant variables and constraints, creating several smaller “sub-problems”. These sub-problems are evaluated with a range of multiple metaheuristic algorithms, with a merge algorithm being applied to determine the place where a large majority of the algorithms agree on a solution. If enough of the smaller metaheuristics agree on which squares should be mined, then this is likely a good solution, being shown that probably there is no need to more computing time to change the final results. However, if a majority of the sub metaheuristics have different conclusions, then a new area in the sample space is explored using SA. Once enough of the sub-metaheuristics agree on a solution, the blocks are mined, and a new area in the sample space is explored for the next time period.

The main goal of this paper is to evaluate the perforemannce of two different categories of metaheuristics for solving CPIT. Along with the main research focus, more specific research questions were created as follows:

1) Which algorithm finds the best solution for CPIT between the SA and the GA using the datasets from *Minelib* [11]?

2) Between the SA and the GA, which algorithm finds a relevant solution to CPIT more quickly?

III. METHODOLOGY

The following subsections present the implementation of the SA and the GA for CPIT, which were presented in [15]. For the performed experiments, CPIT is represented as a MIP with predefined variables and relations, and an objective function that needs to be maximized.

A. Simulated Annealing

The SA algorithm is a heuristic method used to find the global optimum of an objective function. SA algorithm finds the global optimum by searching a solution space containing both local maxima and minima. The SA is a meta-metaheuristic used for global optimization in a large solution space. As a result, for mixed-integer programming problems, which require finding the global optimum, an approximate solution from a SA metaheuristic may be preferred over exact solutions from systematic mathematical methods, such as branch and bound. A SA algorithm works by comparing iterative outputs with current and neighboring objective nodes. If a neighboring node generates a better solution than the current iteration, the neighboring weights are used for the next iteration. One of the objectives of this study is to implement a SA algorithm as a trajectory algorithm, so CPIT can be solved and compared with a well-known Population-based algorithm, which in this case was the GA. From Bertsimas and Tsitsiklis [16], some basic elements of a SA algorithm were implemented into Python from scratch, without any external libraries or packages, except for NumPy (it used its mathematical exponent and random functions). The pseudo-code for the used the SA algorithm is as follows (Algorithm 1):

Algorithm 1. SA algorithm implementation for the CPIT.

```
state = random_start where random_start ∈ solution_space

cost = the cost of the CPIT objective

for i = 1 to number of blocks do
    a)  temp = max(0.01, (i ÷ number of blocks))
    b)  next_state = get neighbor from state’s precedences
    c)  next_cost = get cost of next_state
    d)  acceptance_prob = $exp[-\frac{i}{temp} * (\frac{nextstate - state}{nextstate})]$
if acceptance_prob > rand_int(0, 1) do
        i.  state = next_state
```

```
            ii.    cost = next_cost
 return [list(states), list(costs)]
```

B. *Genetic Algorithm (GA)*

A GA is a metaheuristic search algorithm that mimics the process of natural evolution, belonging to a larger class of evolutionary algorithms. These algorithms generate solutions to optimization problems using several techniques, namely, inheritance, mutation, selection, and crossover, all of them inspired by natural evolution.

Alipour et al. implemented a GA [17]using the CPIT objective function to evaluate the population's fitness. The initial population is randomly generated and afterward normalized, ensuring that the solutions fit the constraints; 200 iterations are run on the population, by using the best solutions at the end, regardless of their fitness. Following this paper, a GA was implemented using Python 3, without any imported external libraries. The pseudo-code which represents the internal logic of the algorithm can be seen here (Algorithm 2):

Algorithm 2. GA implementation for the CPIT.

```
generate an initial population of random solutions

normalize the population

for i = 1 to 200 do
    a)  randomly choose 2 parents P1 and P2
    b)  crossover them to generate offspring_1 and
        offspring_2
    c)  normalize offspring
if offspring_1.fitness > P1.fitness do
            i.    P1 = offspring_1
if offspring_2.fitness > P2.fitness do
            ii.   P2 = offspring_2
return solution with the highest fitness
```

C. *Finding the Maximum Upper Bound*

Using A Mathematical Programming Language (AMPL) [18], it is possible to create mathematical models for optimization problems, and more importantly, MIP problems that contain constraints. AMPL also has a Python compatibility package, which is known as AMPLPy. Using AMPLPy, it is possible to run AMPL model files in Python. Consequently, it is possible to run the CPIT model file in Python to find the maximum upper bound for a designated dataset.

The AMPL model for CPIT has some differences when compared to the proposed Python implementation of CPIT in this paper. While the AMPL model is designed to find the exact solution, the proposed Python implementation of CPIT is used as the objective function to generate neighboring solutions for both the SA and the GA.

D. *Datasets*

All the used datasets in these experiments come from *Minelib* [11]. For CPIT, the number of blocks in the mine and the precedences for each block are known. In this work, three datasets were used: Newman 1, Zuck_small, and KD, with time periods of 6, 20, and 12, respectively. Figure 2 presents the number of blocks and precedences of the three datasets.

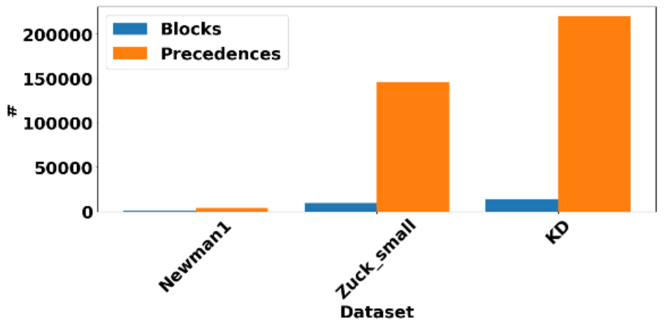

Figure 2. Number of blocks and precedences of each dataset.

IV. RESULTS AND DISCUSSION

For the experiments performed during this work, all the algorithms were run on a single desktop computer with an AMD Ryzen 3950x 3.9 GHz CPU. Doing so made it possible to keep consistent runtimes between each algorithm. All the results shown in Figures 3-5 are of arbitrary profit units, having no consistent way to track the exact amount of profit from an open-pit mine, since the value of resources changes on a daily basis. In section A, the experimental results have been described in detail.

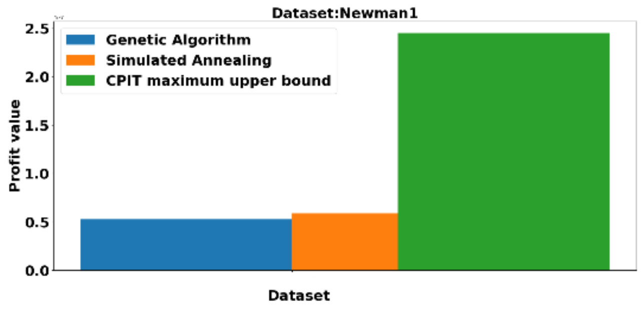

Figure 3. Profit values gained for the Newman1 dataset using the different algorithms.

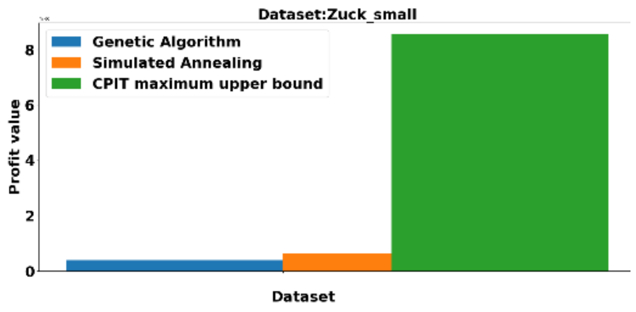

Figure 4. Profit values gained for the Zuck_small dataset using the different algorithms.

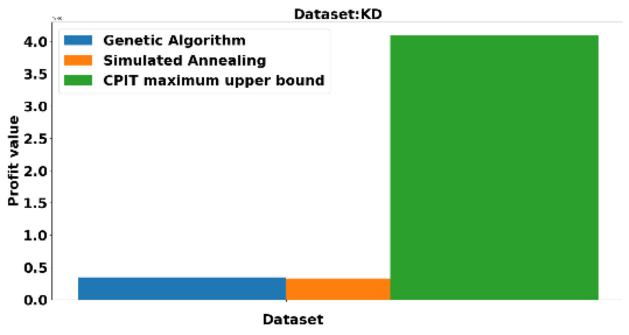

Figure 5. Profit values gained for the KD dataset using the different algorithms

A. THE EXPERIMENTAL RESULTS

The implemented GA was significantly faster in the generation of a result when compared to the SA algorithm. However, overall, both metaheuristic algorithms were rather slow since each of them took several minutes to generate results for the smaller datasets. As the datasets increased, the number of blocks and precedences, the algorithms became even slower. The maximum upper bound for CPIT was attained using an AMPL model of CPIT, and running the AMPL model with the associated dataset. With this, it was possible to determine the maximum solution for CPIT, allowing for the use of a base to the results of the metaheuristic models against the expected maximum solution of CPIT.

The results achieved from the SA and the GA showed that each algorithm took several minutes to complete, even for tiny datasets. There are several reasons for these results, with the most likely being the way Python was used for the implementation of CPIT and the method of finding appropriate random and neighboring solutions.

There are several ways to improve the CPIT model. Since the model was implemented by using lists and arrays for holding data from the datasets, many of the functions to access blocks' data, including the profit value for that block or the precedents of that block, require linear iteration through the array until the correct index is found. As a result, accessing block's data that is needed to generate a neighbor would result in a function that runs in $O(n^2)$ time complexity, since the block index needs to be accessed, as well as the precedence index for that particular block. One crucial step that can be used to reduce runtimes would be to make the CPIT model a multithreaded one, which would allow the model to have access to more than one CPU thread during its execution. For most CPUs, including the one on which the tests were run, implementing a multithreaded capacity would have considerably reduced the runtime of the CPIT model.

V. CONCLUSION

In this paper, two well-known metaheuristics, the SA and the GA, from two different categories, namely, Trajectory and Population-based algorithms, respectively, were applied. The results showed both algorithms lead to the same profit values for Newman1 and KD datasets, while the SA possesses a better profit value when using zuck_small dataset. Improving the implementation of both the SA and the GA, combined with a better optimized CPIT model, may lead to higher quality results that come within a ~5% range of the theoretical upper bounds.

One of the future objectives is to improve the implementation by solving some of the problems of the CPIT model, particularly the time complexity of the algorithm, as well as handling the constraints of the CPIT model in a different manner, where the scheduling of neighboring blocks is more confined to the precedences. Moreover, it is worth noting that the runtime of these algorithms depends on the choice of parameter settings; in this paper, the default settings have been conducted, and it is suggested for future work that different operator settings are addressed. It is also an objective to look at other solutions to CPIT, such as Kenny et al.'s merge search [19] [14], where the SA is applied to generate the neighboring solutions rather than to solve the entire CPIT. From this information, it may be possible to combine metaheuristic algorithms to form only one result instead of implementing separate metaheuristics to find the global optimum.

REFERENCES

[1] A. H. Gandomi, A. Emrouznejad, and I. Rahimi, "Evolutionary Computation in scheduling: A scientometric analysis", Evolurionary Computation in Scheduling, pp.1-10,2020.

[2] R. Behmanesh, I. Rahimi, and A. H. Gandomi, "Evolutionary many-objective algorithms for combinatorial optimization problems: a comparative study," *Archives of Computational Methods in Engineering*, vol. 28, no. 2, pp. 673–688, 2021.

[3] I. Rahimi, A. Ahmadi, A. F. Zobaa, A. Emrouznejad, and S. H. E. A. Aleem, "Big data optimization in electric power systems: A review," *Big Data Analytics in Future Power Systems*, pp. 55–84, 2018.

[4] A. D. Mwangi, Z. Jianhua, H. Gang, R. M. Kasomo, and M. M. Innocent, "Ultimate pit limit optimization methods in open pit mines: A review," *Journal of Mining Science*, vol. 56, no. 4, pp. 588–602, 2020.

[5] C. Meagher, R. Dimitrakopoulos, and D. Avis, "Optimized open pit mine design, pushbacks and the gap problem—a review," *Journal of Mining Science*, vol. 50, no. 3, pp. 508–526, 2014.

[6] E. Ben-Awuah, O. Richter, T. Elkington, and Y. Pourrahimian, "Strategic mining options optimization: Open pit mining, underground mining or both," *International Journal of Mining Science and Technology*, vol. 26, no. 6, pp. 1065–1071, 2016.

[7] É. Lèbre, G. Corder, and A. Golev, “The role of the mining industry in a circular economy: a framework for resource management at the mine site level,” *Journal of Industrial Ecology*, vol. 21, no. 3, pp. 662–672, 2017.

[8] M. Brueckner, A. Durey, R. Mayes, and C. Pforr, “The mining boom and Western Australia’s changing landscape: Towards sustainability or business as usual?,” *Rural Society*, vol. 22, no. 2, pp. 111–124, 2013.

[9] I. Rahimi, A. H. Gandomi, K. Deb, F. Chen, and M. R. Nikoo, “Scheduling by NSGA-II: review and bibliometric analysis,” *Processes*, vol. 10, no. 1, p. 98, 2022.

[10] I. Rahimi, A. H. Gandomi, F. Chen, and E. Mezura-Montes, “A Review on Constraint Handling Techniques for Population-based Algorithms: from single-objective to multi-objective optimization,” *arXiv preprint arXiv:2206.13802*, 2022.

[11] D. Espinoza, M. Goycoolea, E. Moreno, and A. Newman, “MineLib: a library of open pit mining problems,” *Annals of Operations Research*, vol. 206, no. 1, pp. 93–114, 2013.

[12] D. Bienstock and M. Zuckerberg, “Solving LP relaxations of large-scale precedence constrained problems,” in *International Conference on Integer Programming and Combinatorial Optimization*, 2010, pp. 1–14.

[13] A. Kenny, X. Li, and A. T. Ernst, “A merge search algorithm and its application to the constrained pit problem in mining,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 316–323.

[14] A. Kenny, X. Li, A. T. Ernst, and Y. Sun, “An improved merge search algorithm for the constrained pit problem in open-pit mining,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 294–302.

[15] M. Samavati, D. Essam, M. Nehring, and R. Sarker, “A new methodology for the open-pit mine production scheduling problem,” *Omega (Westport)*, vol. 81, pp. 169–182, 2018.

[16] D. Bertsimas and J. Tsitsiklis, “Simulated annealing,” *Statistical Science*, vol. 8, no. 1, pp. 10–15, 1993.

[17] A. Alipour, A. A. Khodaiari, A. Jafari, and R. Tavakkoli-Moghaddam, “A genetic algorithm approach for open-pit mine production scheduling,” *International Journal of Mining and Geo-Engineering*, vol. 51, no. 1, pp. 47–52, 2017.

[18] R. Fourer, D. M. Gay, and B. W. Kernighan, “A modeling language for mathematical programming,” *Management Science*, vol. 36, no. 5, pp. 519–554, 1990.

[19] A. Kenny, X. Li, and A. T. Ernst, “A merge search algorithm and its application to the constrained pit problem in mining,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 316–323.