Evolutionary Approach for Large-Scale Mine Scheduling

Saber Elsayed *[a], Ruhul Sarker [a], Daryl Essam [a], Carlos A. Coello Coello[b]

[a]*School of Engineering and Information Technology, University of New South Wales Canberra, Australia (e-mail: {s.elsayed, r.sarker, d.essam}@unsw.edu.au)*
[b]*Departamento de Computación CINVESTAV-IPN (e-mail: ccoello@cs.cinvestav.mx)*

Abstract

Finding the optimum solution for open-pit mine scheduling problems (OPMSPs) is a well-known challenging problem in the mining industry. The existing methodologies for solving large-scale OPMSPs are mostly limited to conventional optimization approaches and heuristics. However, due to the problem's complexity, represented by high-dimensionality, and hard and soft constraints, the current approaches face challenges in finding good quality solutions with a reasonable computational cost. As an alternative approach, in this paper, we propose an evolutionary approach, based on differential evolution, with three important features: to deal with high-dimensionality, based on the extraction status of each block, we reduce the number of decision variables (blocks) over the planning horizon; to deal with the complex constrained landscape, we develop a repair mechanism that guarantees feasibility and to enhance the algorithm's performance, we incorporate a local search technique. The experimental results on well-known mine deposits, with up to $112,687$ blocks show that the algorithm has the edge over existing methods to obtain better solutions.

Keywords: Open-pit mine, scheduling, evolutionary algorithms, differential evolution

---

1. Introduction

In the mining industry, to extract natural materials from an open pit, a deposit is divided into a three-dimensional grid of fixed-size blocks (i.e., $50 \times 50 \times 20$ ft for the p4hd Gold/Copper mine in Nevada, USA [18]), each with its own characteristics, such as tonnage, grade (proportion of useful mineral in one unit (tonnage). The most common metric used to represent the grade of an ore is grams per tonne (g/t)) and density. Tonnage is predicted by spatial interpolation techniques, i.e., kriging and inverse distance weighting methods [18]. The blocks are categorized either as ore (those with an expected revenue that exceeds their processing expense) or as waste blocks. Here, the goal is to find the optimal long-term planning sequence of blocks that maximizes net profit while satisfying a set of physical requirements (for example, a safe working width for equipment) and operational constraints (for example, a block cannot be extracted before its predecessor blocks, in other words, the extraction process starts at the top of the ore body and progress downward) [33, 24, 36, 20, 34]. For a small number of blocks, the problem can be firstly formulated as an integer linear programming model and then the developed model can be solved using standard optimization solvers. However, in most real-world mine scheduling problems, too many variables and constraints are

---

[*] Corresponding author

involved. As reported in the literature, one such problem contains more than 2 million decision variables and 100 million constraints [18]. This mine scheduling decision problem is known as the open pit mine scheduling problem, which we will refer to as 'OPMSP' in this paper.

Over time, several approaches for solving OPMSPs have been proposed. Most of these approaches were mainly based on integer linear programming (ILP) techniques [8] and/or heuristics for specific cases [22]. For many practical scale OPMSPs, ILP based approaches might be computationally expensive; for instance, the commercial solver CPLEX cannot solve many of these large-scale problems, due to memory overflow or unacceptable computational effort [31], on top of that such commercial solvers are expensive.

On the other hand, although meta-heuristics (such as evolutionary algorithms (EAs) (i.e., genetic algorithms (GAs) [12] and differential evolution (DE) [40]) and swarm intelligence (SI) (i.e., ant colony optimization (ACO) [39] and particle swarm optimization (PSO) [28])) which are population-based stochastic search methods, have demonstrated great success in solving complex optimization problems, research on using them to solve OPMSPs is scant.

Motivated by the abovementioned fact and to take advantage of the flexible nature of EAs, in this paper, a DE algorithm for solving OPMSPs is introduced. Note that DE is very efficient for solving optimization problems in the continuous domain but OPMSPs are integer problems. To take advantage of this, we perform the search process in the continuous domain and then generate integer solutions using a heuristic. To the best of our knowledge, this is the first attempt to adapt DE to solve such problems. In the proposed algorithm, problem dimensionality is reduced over the time horizon, that is, DE is used to solve the model over only one time period. Once the optimization process of a time period is completed, all the blocks mined are removed from the model and then DE optimizes the new model by considering only blocks that have not yet been mined. This process continues until all periods are covered, or all the blocks are mined, in the given planning horizon. In addition, a repair mechanism is carried out to convert infeasible solutions (those do not satisfy the precedence and resources constraints) to feasible ones, with a local search mechanism being adopted to enhance the performance of the proposed methodology.

The performance of the proposed approach is evaluated by solving a set of test instances with up to $112,687$ blocks that are available in the mine library (MineLib) [18]. Compared to existing methods, the results obtained by this approach are better. The performance of the proposed algorithm and the choices of its parameters are analyzed to offer in-depth understanding of its strengths and weaknesses.

The rest of this paper is organized as follows: the problem model is introduced in the following section, followed by a brief description of DE and an overview of related work in Section 3. Then, the proposed algorithm is described in Section 4; and the experimental results and conclusions are discussed in Sections 5 and 6, respectively.

2. OPMSP Model

Mathematically, OPMSPs can be formulated as follows:

$$\text{Maximize } f = \sum_{t=1}^{T} \sum_{b=1}^{B} \hat{p}_{bt} x_{bt}$$

subject to

$$\sum_{\tau \leq t} x_{b\tau} \leq \sum_{\tau \leq t} x_{b'\tau}, \, \forall b \in B, \, b^{'} \in B_b, \, t \in T$$

$$\sum_{t=1}^{T} x_{bt} \leq 1, \, \forall b \in B,$$

$$\underline{L}_{r,t} \leq \sum_{b=1}^{B} q_{br} x_{b,t} \leq \overline{U}_{r,t}, \, t \in T, \, r \in \Re$$

$$x_{b,t} \in \{0,1\}, \, b \in B, \, t \in T \tag{1}$$

where T is the number of time periods in the planning horizon, B the total number of blocks, B_b the set of predecessor blocks b' for block b, q_{br} the amount of operational resource r required to extract the b^{th} block, $\underline{L}_{r,t}$ and $\overline{U}_{r,t}$ the lower and upper bounds of r at time t, respectively, $\Re$ the number of resources, $\hat{p}_{bt}$ the profit obtained from extracting block b at time period t. $x_{bt} = 1$ if b is extracted in time period t, 0 otherwise. The objective is to maximize the net present value of the extracted blocks over the life of the mine. Constraint 1 ensures that no block is mined before its predecessors. Constraint 2 ensures that each b can be extracted only once. Constraint 3 guarantees that the operational resources used in each time period, must be within the upper and lower limits of those resources.

3. DE and Review

This section starts with a brief description of DE followed by a review of existing conventional approaches and meta-heuristics (EA and SI-based) approaches for solving RCPSPs.

3.1. Differential Evolution: brief overview

DE is a population-based stochastic algorithm that uses a set of operators to guide a set of solutions towards the optimal solution [40]. The algorithmic steps of DE (as shown in Algorithm 1) begin with a set of solutions (also called vectors) , i.e.,($X = \{\overrightarrow{x}_1, \overrightarrow{x}_2, ... \overrightarrow{x}_{PS}\}$) randomly generated within the search domain, where PS is the population size. Subsequently, a mutant population is produced by a DE mutation, where new perturbed vectors are produced. Note that a large and growing body of literature has investigated many variations of the mutation operator, with more information given in [11]. Then, each $\overrightarrow{x}_z$ is recombined (through a crossover operator) with its corresponding mutant vector to generate a trial vector $\overrightarrow{u}_z$, where $z = \{1, 2, ..., PS\}$. For many years, two crossover variants (binomial (also known as uniform or discrete) and exponential (also known as two-point)) have widely been used [16]. Finally, a pairwise comparison between $\overrightarrow{x}_z$ and $\overrightarrow{u}_z$ is performed, with the winning solutions based on their quality. The resultant solutions are considered the next population at the next generation.

3.2. RCPSPs: related work

To alleviate time complexity when solving OPMSPs, past research used to either aggregate blocks into layers [7] or ignore the discrete nature of the block-extraction decisions [41]. Consequently, these approaches have proved to be impractical in many cases. In this section, we provide a brief review on uses of the conventional approaches and meta-heuristics that have been proposed to solve OPMSPs.

Algorithm 1 DE framework

```
Define $PS$, $MaxGen$, $F$, $Cr$, $gen \leftarrow 0$;
Generate random $X$;
Calculate the fitness values $f_z\left(\overrightarrow{x}_z\right)\ \forall z = 1, 2, ..PS$;
while $gen < MaxGen$ do
    $f_{gbest} \leftarrow min(f)$;
    $gen \leftarrow gen + 1$;
    Generate new mutant solutions using a mutation operator;
    Generate new solutions ($u$) using a crossover operator;
    Calculate $f_z^{new}(\overrightarrow{u}_z)\ \forall z = 1, 2, ..PS$;
    for $z = 1, 2, ..PS$ do
        if $f_z$ is better than $f_z^{new}$ then
            $f_z^{new} \leftarrow f_z$
            $\overrightarrow{u}_z \leftarrow \overrightarrow{x}_z$
        end if
    end for
    $f \leftarrow f^{new}$
    $x \leftarrow u$
end while
```

3.2.1. Conventional approaches

In [8], a mixed integer linear programming (MILP) formulation of OPMSPs was solved with a branch-and-cut algorithm embedded with an LP relaxation and mixed integer program. The aim of that work was to deliver a method which explicitly incorporates all constraints in the optimization and is capable of producing good solutions. The method used the CPLEX solver in solving the relaxed LP sub-problems. On test instances with a number of blocks between $6,720$ and $209,664$ and 10 time periods, the algorithm was able to obtain near-optimal solutions.

Bolan et al. [6] first formulated the problem as a MIP, and then aggregated (i.e., variables and/or constraints that are *similar* based on predefined criteria are grouped together into new variables or constraints) were used to schedule the mining process, while individual blocks were used for processing decisions. Also, an iterative disaggregation method was proposed with the model solved by the CPLEX solver. Two test instances were considered with the algorithm showing encouraging results.

Cullenbine et al. [10] proposed a sliding time window heuristic (STWH) to solve OPMSPs. The idea was to solve the model over a time window that covers a smaller number of periods and upon completion, it fixed the those periods' variables to the values found, and moved subsequent periods. The process continues until all periods are covered in the time window. Also, instead of completely ignoring the out periods, the proposed STWH maintained an approximate sub-model in the out periods, along with an exact sub-model in the "window" between the fixed and approximate parts of the overall model. On problems with 15 time periods and $25,620$ blocks, STWH was able to obtain near-optimal solutions, while a commercial software failed to solve it. However, STWH was not able to solve a test instance with $53,668$ blocks.

Martinez and Newman [32] introduced a MIP to schedule long- and short-term productions at LKAB's Kiruna mine located in Sweden. To mitigate the problem's complexity, they proposed a variable elimination approach. Also, they decomposed the objective function either by ore grade or by nature of the deviation. The algorithm was able to attain solutions with about 3-6% of total demand.

Chicoisne et al. [9] introduced a decomposition method for solving OPMSPs, that solves the LP version of OMPSPs with a single capacity constraint in each time period. Then, if the resultant solution is infeasible, it is converted to a feasible one by using a rounding algorithm based on a topological sorting approach. It was reported that the algorithm provides near-optimal solutions in reasonable time. However, the time required to run the LP relaxation was not reported.

Lamghari et al. [30] employed three approaches to reduce the computational time needed to solve OPMSPs. The first one was eliminating variables/blocks which were not mined in the optimal solution. The second was using a Maximum Value Feasible Pit (MVFP) heuristic to generate an initial feasible solution, which was then used by the branch-and-bound algorithm; and the last approach employs a Tailored Lagrangian Relaxation (TLR) mechanism. The algorithm showed encouraging performance on test instances with up to $53,668$ blocks.

Liu and Kozan [31] proposed two graph-based algorithms (GBA-Alg1 and GBA-Alg2) that are based on the network flow and conjunctive graph theories for solving OPMSPs. On 10 test instances, the results showed that GBA-Alg2 was better than GBA-Alg1, but computationally expensive. Unfortunately, if we compare the results obtained, we will infer that its performance was inferior to other approaches.

The authors in [22] suggested a new approach based on aggregation and disaggregation heuristics to solve OPMSPs. First, their approach aggregated blocks to reduce its size and make it computationally tractable then approximately solved the problem using IP techniques (forward stage). Then, in the backward stage, the approach used the solution obtained from the forward stage and fixed some variables in the original problem. Although the algorithm was only compared against two state-of-the-art approaches, the results were better for 9 problems out of 11. The algorithm was then extended in [21] and showed good results.

Samavati et. al. [37] proposed a heuristic for solving OPMSPs. Firstly, they added some inequality constraints to the LP relaxation formulation and then used the LP solutions to generate efficient weights for the sequencing heuristic. On 11 test instances, the algorithm showed better performance than those of other heuristics. Although the computational time of that algorithm was good, it was not clear if the authors considered the time consumed to solve the mixed integer problem or not. The same authors extended their work by proposing a new method called adding cut (AC). Firstly, they iteratively added some special cuts to the LP relaxation formulation and solved it using the CPLEX solver, with the resultant solution improved by a topological sort heuristic. The algorithm was tested on a set of instances taken from MineLib and showed improvements in the quality of solutions for 6 problems out of 11, in comparison with two other existing methods.

Kenny et al. [24] proposed a merge search algorithm, which uses information from a population of solutions to create a smaller sub-problem which can then be used by the MIP solver. The algorithm was evaluated on 6 OPMSPs and showed its superiority to earlier work introduced by the authors [25, 23], but the algorithm still could not compete with other existing algorithms.

3.2.2. EAs-based approaches

Although EAs have been used to solve OPMSPs for more than 20 years [13] when Denby and Schofield used a traditional GA to solve such problems, only very limited subsequent research has been conducted.

Myburgh and Deb [33] proposed a GA for solving OPMSP, called evORElution. In it, an integer-based representation was used to encode chromosomes, with a sequencer used to generate feasible solutions. To search for the optimum schedule, evORElution tried to minimize the average waste material violation across all periods. After a binary tournament selection has taken place, one of two crossover operators was used to generate new offspring, i.e., if released stages were known, a single-point crossover would be carried out; otherwise a period crossover operator would be used. For the period undergoing mutation, a new sub-sequence of blocks was generated. The algorithm kept running until the objective values had not been improved for 10 consecutive generations. It is worthy to mention that the authors used a master-slave parallel implementation. evORElution was tested on two instances: (1) $470,161$ blocks divided into 40 periods; and (2) $62,761$ blocks with 32 periods, but suffered from a rapid loss of diversity. The authors improved the algorithm by optimizing a single period at a time, and once it was completed, it moved to the next one, and so on. Unfortunately, no details about the number of runs used and comparisons with other methods on other complex instances were given.

A hybridization of maximum flow and a GA based on linear programming was introduced in [35], in which GA was used to control the flow in the arcs to the maximum flow algorithm to general the schedule. The results showed that the performance of the algorithm was deteriorated as it did not effectively control the extraction of more waste blocks during later periods.

In [29], the authors provided a comparative study of a small number of DE parameters and operators on only 3 mining problems, with the results showing using $cr = 0.1$, $F = 0.4$ is recommended, while no single DE operator was performing the best in all three problems. Unfortunately, the study neither analyzed the algorithm in a well-known test suit, nor compared it to other well-known algorithms.

3.2.3. SI-based approaches

Shishvan and Sattarvand [39] introduced an ACO for the solution of a real Copper–Gold deposit with just over 350,000 blocks. The algorithm used a penalty approach to merge the constraints into the objective function. Also, at every generation, only the global best ant was allowed to add pheromones, with the evaporation process only applied to the blocks of the global best schedule, not to all the ants. The results showed that the proposed ACO was able to enhance quality by 12% in comparison to traditional algorithms.

Ferland et al. [19] formulated OPMSPs as a resource-constrained project scheduling problem (RCPSP), where the representation was based on a priority value encoding. After the generation of the initial solutions (swarm), the swarm was evolved using PSO. The algorithm was tested on 20 randomly generated small-scale problems, over a two dimensional grid with 20 layers and 60 blocks wide. Although the algorithm showed encouraging results, its performance on well-known large-scale mine scheduling problems is not known.

Flowing their attempt to solve a simple version of OPMSPs [27], in [28], OPMSPs were turned into constraint satisfaction problems and were then solved using the CPLEX solver to get multiple feasible solutions (swarm). This swarm was then updated by a continuous version of a PSO algorithm,while using (1) a repair method to convert infeasible solutions to feasible ones and (2) a back transform scheme to determine the period in which a block was scheduled. Unfortunately, the algorithm was only tested on a hypothetical 3D block model, which made it difficult to provide in-depth understanding about the pros and cons of the algorithms. The algorithm was then extended by including a bat algorithm to solve one uncertain OPMSP with grade uncertainty [26].

4. DE for OPMSPs

In this section, the proposed DE algorithm, denoted as DE-OPMSPs, is discussed and its steps are given in Algorithm 2. To smooth the description of the proposed approach, it will be helpful to give the meaning of the variables below.

variable	meaning
B_Υ	not mined blocks
x^{cont}	real-valued population
$\overrightarrow{x}_z^{cont}$	the z^{th} real-valued solution in x^{cont}
x^{int}	integer population
$x_{z,b}$	the z^{th} mining status vector $\in \{0,1\}$, where 1 means block b is mined and 0 is not
$x^{int,fes}$	feasible integer population
$x_z^{int,fes}$	the z^{th} integer solution in $x^{int,fes}$
$\overrightarrow{u}_z^{cont}$	the z^{th} trial real-valued solution generated by DE mutation and crossover operators
$\overrightarrow{u}_z^{int}$	integer trial solution

To alleviate the complexity of the problem, the model is simplified and iteratively solved over a time window that covers only one period until all periods in the time horizon are covered. Meaning, for every time period (t) the problem is reformulated as follows:

$$\text{Maximize}\, f_t = f_{t-1} + \sum_{b=1}^{B_\Upsilon} \hat{p}_{bt} x_b$$

subject to

$$x_b \leq x_{b'} \;\forall b \in B_\Upsilon,\; b^{'} \in B_b$$

$$\underline{L}_{r,t} \leq \sum_{b=1}^{B_\Upsilon} q_{br} x_b \leq \overline{U}_{r,t},\; r \in \Re$$

$$x_b \in \{0,1\},\; b \in B_\Upsilon \tag{2}$$

Algorithm 2 General framework of DE-OPMSPs

```
Define $PS$, $\alpha$, $MaxGen$, $\overrightarrow{x}_{P,b} \leftarrow T+1$, $\overrightarrow{x}_{S,b} \leftarrow 0$, $sim \leftarrow 0$, and all other parameters required (Section 5);
Generate random $x^{cont}$ and their corresponding $x^{int}$;
Convert $x^{int}$ to feasible solutions ($x^{int,fes}$), update their $x^{cont}$ and generated $x$ (Section 4.2);
Calculate the fitness values $f_z\left(\overrightarrow{x}_z\right)$ $\forall z = 1, 2, ..PS$;
$f_{gbest} \leftarrow min(f)$;
for $t = 1 : T$ do
  while $gen < MaxGen$ && $sim < \alpha$ do
    $gen \leftarrow gen + 1$;
    Generate new solutions ($u^{cont}$) (equation 3);
    Convert $u^{cont}$ to new feasible integer-based solutions and update $x^{cont,new}$ and $x^{new}$ (Section 4.2);
    Calculate the fitness values for $x^{new}$;
    Apply selection operator and update $x$, $x^{cont}$, $x^{int,fes}$ and $f$;
    $f_{best,gen} \leftarrow min(f)$;
    if $f_{best,gen} == f_{best,gen-1}$ then
      $sim \leftarrow 1$;
    else
      $sim \leftarrow 0$;
    end if
  end while
  Apply local search and update $f_{best,gen}$
  Update $\overrightarrow{x}_{P,b}$, $\overrightarrow{x}_{S,b}$ and $B_{\Upsilon}$;
  $f_{gbest} \leftarrow f_{gBest} + f_{best,gen}$;
end for
```

where f_{t-1} is the best objective value obtained at $t-1$ and with $f_{(t=0)-1} = 0$. B_{Υ} is the list of all blocks that have not been mined yet, which means that at $t = 0$, $B_{\Upsilon} = B$ and at the end of the optimization process of each time period, B_{Υ} is updated by removing any block with $x_b = 1$. All the other symbols are discussed in Section 1.

Note that this model represents the same problem described by the model in Section 2, in which operational constraints, i.e., slope constraints (each block cannot be mined before its predecessors), and resource constraints, are considered.

To begin, two reference vectors ($\overrightarrow{x}_S \in \{0,1\}$ and $\overrightarrow{x}_P$) of length B are initialized, such that $x_{S,b} = 0$ (0 means the current block has not been mined yet) and $x_{P,b} = T+1 \ \forall\, b = 1, 2, ..., B$. The value of $\overrightarrow{x}_{S,b}$ will change to 1 and $\overrightarrow{x}_{P,b}$ to t, if a block b is extracted at time period t. Note that these two vectors are updated at the end of the entire optimization process of each time period.

Regarding the optimization process, at time period $t = 0$, a real-valued initial population (x^{cont}) of size PS is randomly generated, with each $x^{cont}_{z,b} \in [0,1]$, $\forall\, z = \{1, 2, ..PS\}$ and $b = 1, 2, ..., B_{\Upsilon}$. Then, feasible integer solutions $\left(x^{int,fes}\right)$ and mined blocks ($x_{z,b} \in \{0,1\}$) are generated with their corresponding continuous vectors (x^{cont}) updated (see Section 4.1). The quality (objective value) of each solution ($f_t\left(\overrightarrow{x}_z\right)$) is then measured.

Subsequently, for every $\overrightarrow{x}^{cont}_z$, a new trial continuous vector ($\overrightarrow{u}^{cont,new}_z$) is generated using the current-to-p-best mutation and binomial crossover, as

$$u_{z,b}^{cont} = \begin{cases} x_{z,b}^{cont} + F_z \left(x_{pbest,b}^{cont} - x_{z,b}^{cont} + x_{r_1,b}^{cont} - x_{r_2,b}^{cont} \right) \\ \qquad\qquad\qquad if\, (rand \leq cr_b \,\text{or}\, b = b_{rand}) \\ x_{z,b}^{cont} \qquad\qquad\qquad\qquad \text{otherwise} \end{cases} \tag{3}$$

where $r_1 \neq r_2 \neq z$ are random integer numbers, with $\overrightarrow{x}_{r_1}^{cont}$ and $\overrightarrow{x}_{r_2}^{cont}$ randomly selected from x^{cont} [43], b_{rand} randomly selected from B_{Υ} and $\overrightarrow{x}_{pbest}^{cont}$ randomly selected from the *pbest* solutions in x^{cont}. The rationale of using such a DE mutation is its efficiency in solving many optimization problems [42, 15, 17, 14].

Based on the values of each $\overrightarrow{u}_z^{cont}$, a corresponding integer-based solutions $\left(\overrightarrow{u}_z^{int}\right)$ is generated (see Section 4.1).As a $\overrightarrow{u}_z^{int}$ may be infeasible, the following two steps are carried out, with the first probabilistically applied:

1. For every $\overrightarrow{u}_z^{int}$, a random number $\in [0, 1]$ is generated, and if it is less than a predefined probability $(Prob_{prec})$, $\overrightarrow{u}_z^{int}$ is updated to represent a sequence of blocks that satisfies the precedence constraints (step one in Section 4.2).
2. Using $\overrightarrow{u}_z^{int}$, a new solution $\overrightarrow{x}_z^{new,int,fes}$ that satisfies the precedence and resources constraints is generated, with its corresponding $\overrightarrow{x}_z^{cont,new}$ and $\overrightarrow{x}_z^{new}$ being produced (see Algorithm 3).

Once a feasible solution is generated, its objective value $(f_t\,(\overrightarrow{x}_z^{new}))$ is calculated and then a selection operator takes place to keep the fittest ones in the next generation, i.e., update x, x^{cont} and x^{int}. The optimization process continues until at least one of the following stopping criteria is met: (1) $MaxGen$ generations are carried out, or (2) the best fitness value is not improved for α consecutive generations.

After that, a local search, discussed in section is sub-section 4.3[38, 1], is applied to the best solution $(\overrightarrow{x}_{best})$ and if the new fitness value $(f_t\,(\overrightarrow{x}_{best,LS}))$ is better than $f_t\,(\overrightarrow{x}_{best})$, $\overrightarrow{x}_{best,LS}$ replaces $\overrightarrow{x}_{best}$.

Finally, $\overrightarrow{x}_{S,b}$ and $\overrightarrow{x}_{P,b}$ $\forall b = 1, 2, ..., B$ are updated, such that $\overrightarrow{x}_{S,b} = 1$ and $\overrightarrow{x}_{P,b} = t$ for all $\overrightarrow{x}_{best} = 1$. Remember, B_{Υ} is also updated by removing any mined block, i.e., $x_b = 1$. The process continues until all the time periods are covered or all the blocks are extracted.

4.1. Representation

As in this paper, an OPMSP is represented by an integer-based mathematical model, while the DE uses real-valued solutions, it is essential to consider two representations. As previously mentioned, PS real-valued solutions (x^{cont}) are randomly generated $(x^{cont} = \{\overrightarrow{x}_1^{cont}, \overrightarrow{x}_1^{cont}, ... \overrightarrow{x}_z^{cont}, ..., \overrightarrow{x}_{PS}^{cont}\})$, with each $x_{z,b}^{cont} \in [0, 1]$, $\forall\, z = \{1, 2, ..PS\}$ and $b = 1, 2, ..., B_{NotMined}$. Then, every $\overrightarrow{x}_z^{cont}$ is converted to a vector of integer values $(\overrightarrow{x}_z^{int})$ by sorting the values in $\overrightarrow{x}_z^{cont}$ in an ascending order and then $\overrightarrow{x}_z^{int}$ is considered the place of each block in the order. To clarify, assuming there is a vector with 10 blocks, Figure 1 shows how to convert $\overrightarrow{x}_z^{cont}$ to $\overrightarrow{x}_z^{int}$.

4.2. Generating feasible solutions

To satisfy the precedence constraints, a repair method is used. It is worth mentioning that to save the computational effort, the following iterative processes take place on only the blocks which have not been extracted.

block	1	2	3	4	5	6	7	8	9	10
$\overrightarrow{x}^{cont}_z$	0.11	0.45	0.85	0.49	0.75	0.19	0.25	0.05	0.67	0.99

$\Downarrow$sort

block	8	1	6	7	2	4	9	5	3	10
$\overrightarrow{x}^{cont}_z$	0.05	0.11	0.19	0.25	0.45	0.49	0.67	0.75	0.85	0.99

$\Downarrow$final

$\overrightarrow{x}^{int}_z$	8	1	6	7	2	4	9	5	3	10

Figure 1: Conversion of real-valued solution to integer-based one

First, blocks and their precedence relationships are represented as a directed acyclic graph (DAG), i.e., every directed edge ij from node i to node j means that j depends of i. The degree of each node is represented by the number of predecessors, i.e., a node with no predecessors has a 0 degree. Then, based on the sequence represented in $\overrightarrow{x}^{int}_z$, an iterative process is carried out, that is selecting a block with a degree of 0, then removing it and its outgoing edges from the DAG and subsequently updating the degree of the remaining blocks. The process continues until all blocks are selected, hence $\overrightarrow{x}^{int}_z$ is updated.

Subsequently, the steps in Algorithm 3 take place. For every block $x^{int}_{z,b}$, if its predecessors have been extracted and there are enough resources, then the block is marked as extracted ($x_{z,b} = 1$, line 7) at time period t, and $x^{cont}_{z,b}$ and the remaining resources are updated (line 7 and 8, respectively), otherwise we move to the next block in $\overrightarrow{x}^{int}$.

Once all blocks are checked, to update every $x^{cont}_{z,b}$ for block which has not been extracted, another cycle goes through every $x_{z,b} = 0$, with its $x^{cont}_{z,b}$ assigned a real value bigger than that of the last block mined (see line 16). The worst time complexity for these steps should be $O(n)$. Finally, $\overrightarrow{x}^{cont}_z$ is sorted in ascending order and then $\overrightarrow{x}^{int}_z$ is updated as the place of each block in the order (line 13). As we are using the quick sort algorithm, the time complexity will be $O(nlogn)$. Therefore, the overall time complexity for this algorithm should be $(O(n)+O(nlogn))$. By focusing only on the dominant term, the time complexity is $O(nlogn)$.

It is important to note that the first step mentioned above is done for all the initial solutions, while it is probabilistically applied to any new solution generated by DE. The intuitive reason for this is to reduce the computational time over generations and it is expected that the possibility to generate feasible sequences increases with the progression of the evolutionary process.

To illustrate the abovementioned steps, assume that we have an open-pit with 5 blocks, as shown in Fig. 2, where each node represents the block *id* (expected profit, extraction cost) and the available budget in every time period is 6K. Starting with an initial random $\overrightarrow{x}^{int} = [1, 2, 4, 3, 5]$, the first step is to make sure that this solution satisfies the precedence constraints by applying a topological sort that can generate different feasible variations. For simplicity, assume that the new $\overrightarrow{x}^{int}$ is $[1, 2, 3, 4, 5]$. Once it is generated, the steps in Algorithm 3 take place, $\overrightarrow{x}$ is initialized to $[0, 0, 0, 0, 0]$ and *counter* is set to 1. Then, we check if x^{int}_1 can be extracted or not. In this example, the extraction cost for x^{int}_1 is 3K which is less than the budget available (6K). Subsequently, x_1 is set to 1 (i.e., $\overrightarrow{x} = [0, 0, 0, 0, 0]$), $x^{cont}_1 = \frac{1+rand\in[0,1]}{5}$ and *counter* is increased by 1, i.e., $counter = 2$. Now, the process comes to x^{int}_2, as the remaining budget (3K) is less than that needed to extract it, no action is taken. Then, we move to the next block in x^{int}, which is 3. x^{int}_3 can be extracted as there is sufficient budget. Subsequently, x_3 is set to 1 ($\overrightarrow{x} = [1, 0, 1, 0, 0]$), $x^{cont}_3 = \frac{2+rand\in[0,1]}{5}$, and $counter = 3$. This sets the remaining budget at $t = 0$ to *zero*. Meaning, neither x_4 nor x_5

Algorithm 3 Generating feasible solutions at time period t

```
Define $B_{\Upsilon}$, $L \leftarrow$ size of $B_{\Upsilon}$, $counter \leftarrow 1$, $\overrightarrow{x_z} \leftarrow 0$;
for $i = 1 : L$ do
    $b \leftarrow x^{int}_{z,i}$;
    if $(x^{P}_{z,b'} \leq t)$ then
        if $q_{br} \leq \overline{U}_{r,t} \;\; \forall r \in \Re$ then
            $x_{z,b} \leftarrow 1$;
            $x^{cont}_{z,b} \leftarrow \frac{couner+rand\in[0,1]}{B}$;
            $\overline{U}_{r,t} \leftarrow \overline{U}_{r,t} - q_{br} \forall r \in \Re$;
            $counter \leftarrow counter + 1$;
        end if
    end if
end for
for $i = 1 : L$ do
    if $x_{z,b} = 0$ then
        $b \leftarrow x^{int}_{z,i}$;
        $x^{cont}_{z,b} \leftarrow \frac{couner+rand\in[0,1]}{B}$;
    end if
end for
Sort $\overrightarrow{x}^{cont}_{z}$ then update the sequence in $\overrightarrow{x}^{int,feas}_{z}$
```

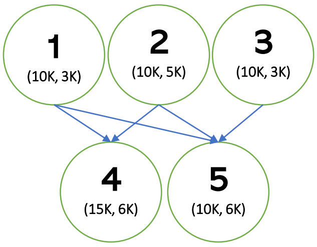

Figure 2: Open-pit example with 5 blocks, each node represents block id (expected profit, extraction cost)

can be extracted. The final step is to update x^{cont}_{b} $\forall b = 1 \ldots 5$ and $x_b = 0$, such that $x^{cont}_{b} = \frac{3+rand\in[0,1]}{5}$.

4.3. Local Search

The local search used is based on a branch-and-bound algorithm [5]. Firstly, a pre-processing step (also called presolve [3] [2]), which refers to a collection of problem reductions that are intended to reduce the size of the problem and to tighten its formulation, is used. The problem is then relaxed by removing the condition that all variables should be integer. Then, it is solved using the simplex method [4]. If the final solution does not satisfy all integer values, then a decision variable that is not integer is selected and then two branches are created (i.e., two sub-problems). The same procedure is used to solve these branches. Note that at one point we may have more than one variable which needs branching. Therefore, selecting the proper variable to branch next is important and is done here by a heuristic [2].

4.4. Adaptation of F and Cr

The methodology introduced in [42] is adopted.

- First, H values for both parameters (M_{Cr}, M_F) are initialized at a value of 0.5.

- Cr_z and F_z are assigned to every $\overrightarrow{x}_z^{cont}$ as

$$Cr_z = \text{randni}(M_{Cr,r_z}, 0.1) \tag{4}$$

$$F_z = \text{randci}(M_{F,r_z}, 0.1) \tag{5}$$

  where r_z is randomly selected from $[1, H]$, and randni and randci values randomly selected from normal and Cauchy distributions with mean values of M_{Cr,r_z} and M_{F,r_z}, respectively, and with variance of 0.1.

- After every *gen*, successful Cr_z and F_z are recorded in S_{Cr} and S_F, respectively, and then M_{Cr} and M_F are updated as

$$M_{Cr,\lambda} = mean_{WA}\,(S_{Cr})\ if\ S_{Cr} \neq \text{null} \tag{6}$$

$$M_{F,\lambda} = mean_{WL}\,(S_F)\ if\ S_F \neq \text{null} \tag{7}$$

  where $1 \leq \lambda \leq H$ is a position in the memory to be updated. λ is initialized to 1 and is then incremented whenever M_{Cr} and M_F are updated, and if it is greater than H, it is set to 1 with $\text{mean}_{WA}(S_{Cr})$ and $\text{mean}_{WL}(S_F)$ computed, respectively, as follows

$$\text{mean}_{WA}(S_{Cr}) = \sum_{\gamma=1}^{|S_{Cr}|} \omega_\gamma S_{cr,\gamma} \tag{8}$$

$$\text{mean}_{WL}(S_F) = \frac{\sum_{\gamma=1}^{|S_F|} \omega_\gamma S_{F,\gamma}^2}{\sum_{\gamma=1}^{|S_F|} \omega_\gamma S_{F,\gamma}} \tag{9}$$

  where $|S_{Cr}|$ is the number of successful Cr recorded in S_{Cr}, $|S_{Cr}|$=$|S_F|$, $w_\gamma = \frac{\Delta f_\gamma}{\sum_{\gamma=1}^{|S_{cr}|} \Delta f_\gamma}$ and $\Delta f_\gamma = |f_{\gamma,t,gen-1} - f_{\gamma,t,gen}|$.

5. Experimental results

The proposed algorithm is tested on 10 problems taken from MineLib [18]. These problems come from real-world mining project and simulated data, as follows: (1) KD is a copper deposit located in North America; (2) P4HD is a gold and copper deposit in North America; (3) W23 and McLaughlin are gold mines located in North America; (4) McLaughlin limit is the final pit of McLaughlin computed by Somrit (2011); (5) Newman1, Marvin, Zuck small, Zuck medium, Zuck large and SM2 are conceptually simulated, with the last one simulated based on a nickel deposit in Brazil. The characteristics of these problems are shown in Table 1.

The initial algorithm's parameters were set as follows: $PS = 30$, $Prob_{prec}$ adaptively reduced from 1.0 to 0.00 (i.e., $= 1.0 - \frac{gen}{MaxGen}$, where $MaxGen = \frac{15{,}000}{PS}$), $\alpha = 30$ consecutive generations and *pbest* was $\frac{PS}{3}$. For adapting F and Cr, the history size (H) was set to 6 with 5 runs carried out. The algorithm was implemented in JAVA, with all the experiments conducted on a PC with a Core(TM) i7-3770 CPU @ 3.40GHz, 8.0 GB RAM and Windows 7.

Table 1: Characteristics of MineLib instances (*noPrCon* number of precedence constraint and *noCon* total number of constraints)

Problem	LP upper bound ($)	T	Blocks	variables	$noPrCon$	$noCon$
Newman	24,176,865	6	1060	6360	3922	10,294
Zuck small	847,013,492	20	9400	188,000	145,640	333,680
KD	409,104,193	12	14,153	169,836	219,778	389,626
Zuck medium	669,493,906	15	29,277	439,155	1,271,207	1,710,392
P4HD	247,405,783	10	40,947	409,470	738,609	1,148,099
Marvin	856,206,143	20	53,271	1,065,420	650,631	1,716,091
W23	399,960,130	12	74,260	891,120	764,786	1,655,942
Zuck large	57,265,626	30	96,821	2,904,630	1,053,105	3,957,795
SM2	1,647,167,503	30	99,014	2,970,420	96,642	3,067,122
McLaughin lim	1,078,662,109	15	112,687	1,690,305	3,035,483	4,725,803

Table 2: Average fitness values and computational time obtained by DE-OPMSPs with different values of PS

Prob.	PS			
	10	20	30	40
	Average fitness values			
Newman	**24,176,864.825**	**24,176,864.825**	**24,176,864.825**	**24,176,864.825**
Zuck small	**847,022,705.562**	**847,022,705.562**	**847,022,705.562**	**847,022,705.562**
KD	409,102,820.239	409,103,817.080	**409,104,124.828**	409,103,313.298
P4HD	**247,405,783.326**	**247,405,783.326**	**247,405,783.326**	**247,405,783.326**
	Time (seconds)			
Newman	**3.0102**	3.9358	3.9682	4.9072
Zuck small	**417.703**	459.146	471.2108	499.7182
KD	**586.818**	603.9118	634.291	646.5824
P4HD	**2878.8582**	3104.2462	3174.8106	3139.3964

5.1. Parametric analysis

In this subsection, three sets of experiments designed to analyze the effects of: (1) PS, (2) α, and (3) $Prob_{prec}$ are described. As the computational time needed to solve all the test instances is huge, this limits our ability to analyze all the parameters. So, in this subsection, these three parameters are analyzed on a subset containing 4 instances (Newman, Zuck small, KD and P4HD). Remember that P4HD has $40,947$ blocks to mine over 10 time periods.

5.1.1. $\boldsymbol{PS}$

In this set of experiments, four variants of DE-OPMSPs were run by varying PS as 10, 20, 30, and 40 with $\alpha = 30$ and $Prob_{prec}$ set as adaptive, as discussed in Section 5.

From the results presented in Table 2, it is clear that changing PS did not have an effect on the performance of DE-OPMSPs in solving Newman, Zuck small and P4HD, while using 30 individuals showed improvement in the quality of solutions for KD. Using the Friedman test, the variants with $PS = 30$ were positioned at the first place with a mean rank of 2.88, followed by $PS = 20$, $PS = 40$ and $PS = 10$ with a mean rank of 2.63, 2.38 and 2.13, respectively. Note that the bigger the mean rank, the better the variant.

Considering the computational time recorded, the results presented in Table 2 demonstrate that DE-OPMSPs with $PS = 10$ was the fastest, although it was not the best based on the quality of solutions. The reason for this might be that it meets the second stopping criterion quickly, i.e., the best solution was not improved for 30 consecutive generations. Over the test instances, DE-OPMSPs with $PS = 20$ came 2^{nd}, while the algorithms with 30 and 40

Table 3: Average fitness values and computational time obtained by DE-OPMSPs with different values of $Prob_{prec}$

Prob.	Average fitness values ($)					
	$Prob_{prec}$					
	adaptive	1.0	0.75	0.5	0.25	0.01
Newman	**24,176,864.83**	**24,176,864.83**	**24,176,864.83**	**24,176,864.83**	**24,176,864.83**	**24,176,864.83**
Zuck small	**847,022,705.56**	**847,022,705.56**	847,022,223.16	847,021,740.76	**847,022,705.56**	**847,022,705.56**
KD	**409,104,124.83**	409,103,636.02	**409,104,124.83**	409,102,494.22	409,102,914.19	409,101,177.10
P4HD	**247,405,783.37**	**247,405,783.37**	**247,405,783.37**	**247,405,783.33**	**247,405,783.33**	**247,405,783.33**
	Time (seconds)					
Newman	3.97	4.11	4.16	3.51	3.39	**2.74**
Zuck small	471.21	710.30	534.49	420.41	289.91	**221.22**
KD	634.29	1232.88	569.27	646.87	577.59	**543.74**
P4HD	3174.81	3700.00	3197.63	3128.55	2781.37	**2650.65**

Table 4: Average mean rank by Friedman test of DE-OPMSPs with 6 different $Prob_{prec}$ values (higher rank means better performance)

adaptive	1.0	0.75	0.5	0.25	0.01
4.25	3.88	3.63	2.5	3.63	3.13

individuals were 3rd and 4th respectively. Because of the time difference between DE-OPMSPs with $PS = 20$ and $PS = 30$, at this stage we preferred the variant with $PS = 30$.

5.1.2. $\boldsymbol{Prob_{prec}}$

This set of experiments fixed PS at the best value found above, i.e., 30, α to 30 consecutive generations and uses 6 different variations of $Prob_{prec}$: (1) adaptive, as previously described, (2) 1.0, (3) 0.75, (4) 0.5, (5) 0.25 and (6) 0.01. The average fitness values, reported in Table 3 demonstrate that $Prob_{prec}$ did not have an effect on the quality of solutions obtained in two test instances (Newman and P4HD). Also, it was found that when the proposed algorithm adaptively controlled $Prob_{prec}$ it was able to obtain the best results for all of the 4 test instances. Also, we ranked all the variants based on the Friedman test, see Table 4, with the results showing that the adaptive version of $Prob_{prec}$ was in the lead with a mean score of 4.25 (remember the biggest value is the best).

Also, what stands out in Table Table 3 is that using a small value of $Prob_{prec}$ reduces computational time, but this affects the quality of solutions. Also, it was noticed that setting $Prob_{prec}$ as adaptive was faster than setting it to 1.0 or 0.75.

Overall, bearing in mind the quality of solutions, average ranking and computational time of each variant, we conclude that setting $Prob_{prec}$ as adaptive is the most preferable approach.

5.1.3. $\boldsymbol{\alpha}$

The third set of analyses examines the impact of α in the performance of the proposed approach. In doing this, PS was set to the value of 30, $Prob_{prec}$ adaptive and α changed as 10, 20, 30 and 40 consecutive generations. The average fitness values presented in Table 5 show DE-OPMSPs with $\alpha = 20$ was the best, though the other three variants were able to obtain similar results for at least two test instances. Surprisingly, the proposed algorithm with $\alpha = 40$ was the worst variant. Based on a Friedman test, setting α at a value of 20 was ranked the best, while both $\alpha = 10$ and $\alpha = 30$ came in 2nd place, and $\alpha = 40$ was 4th.

In line with our expectations, using a small value of α reduces the computational load (see Table 5), but this came with a cost in the quality of solutions. Generally speaking, if the computational time was the only concern, α

Table 5: Average fitness values and computational time obtained by DE-OPMSPs with $\alpha = 10$, 20 , 30 and 40

Prob.	Average fitness values			
	α			
	10	20	30	40
Newman	**24,176,864.825**	**24,176,864.825**	**24,176,864.825**	**24,176,864.825**
Zuck small	**847,022,705.562**	**847,022,705.562**	**847,022,705.562**	847,022,223.161
KD	409,102,860.942	**409,104,686.593**	409,104,124.828	409,104,286.243
P4HD	**247,405,783.326**	**247,405,783.326**	**247,405,783.326**	**247,405,783.326**
	Time (seconds)			
Newman	**2.63**	3.39	3.97	4.46
Zuck small	**247.03**	384.41	471.21	562.24
KD	**580.61**	589.48	634.29	653.81
P4HD	**2733.94**	2975.99	3174.81	3195.40

Table 6: GAP% results obtained by DE-OPMSPs, LR-MTS, SH-VND, LR-MTS-VND, GBA, BAA, GA+MF and LPA

Prob.	LR-MTS	SH-VND	LR-MTS-VND	GBA	BAA	GA+MF	LPA	DE-OPMSPs (Proposed)		
								best	average	std.
Newman	4.1	1.68	2.34	1.38	1.28	1.47	**1.26**	**1.26**	**1.26**	0
Zuck small	7.7	1.8	4.88	2.98	0.86	1.92	2.41	**0.84**	**0.84**	0
KD	3.1	–	–	2.28	0.19	3.39	0.16	**0.10**	**0.10**	4.32E-06
Zuck medium	13.4	8.08	9.04	**5.74**	5.83	–	5.78	5.83	5.83	0
P4HD	0.5	6.61	0.19	0.08	0.2	2.68	0.36	**0.004**	**0.004**	0
Marvin	5	1.87	3.2	1.66	0.89	1.51	**0.67**	0.89	0.89	2.81E-04
W23	2.1	9.96	1.06	0.19	0.55	4.79	0.38	**0.17**	**0.17**	0
Zuck large	1.1	8.07	0.56	12.01	0.24	–	1	**0.19**	0.21	1.12E-02
SM2	0.2	5.85	**0.04**	0.1	0.08	9.18	0.09	0.06	0.06	0
McLaughin lim	0.5	–	–	9.04	0.07	5.28	0.06	**0.03**	**0.03**	0

may be set at 10. On the other hand, if the quality of solutions was the main target, which is the case here, $\alpha = 20$ is a good choice.

5.2. Final results and comparison with state-of-the-art methods

In this subsection, the proposed algorithm with the final set of parameters (i.e., $PS = 30$, $Prob_{prec}$ adaptive and $\alpha = 20$) was run to solve all 10 test instances. The percentage difference between the solution obtained by the algorithm ($fit_{alg,best}$) and the LP upper bound (UP_{LP}), i.e., $GAP\% = \left(\frac{UP_{LP} - fit_{alg,best}}{UP_{LP}}\right) \times 100\%$ were calculated and then the best, average and standard deviation results are shown in Table 6. It is apparent from this table that the proposed method was capable of attaining consistent results, i.e., the standard deviation values were very small.

Turning now to the comparison with the state-of-the-art methods. The results of 6 state-of-the-art algorithms, known as: (1) LP relaxation using a modified TopoSort (LR-MTS) [18]; (2) SH-VND and LR-MTS-VND heuristics [30]; (3) graph based algorithm (GBA) [31]; (4) block aggregation algorithm (BAA) [22]; (5) LP-based algorithm (LPA) [37], (6) GA with maximum flow (GA+MF) [35] are shown in the same Table 6.

Overall, these results revealed that the proposed method outperformed all the other algorithms. For Zuck small, KD, P4HD, W23, Zuck large and McLaughin lim, the proposed algorithm was clearly superior to all the other methods. For the Newman test instance, DE-OPMSPs was able to obtain the best result reported in the literature, which was also obtained by the LPA, but the other algorithms could not achieve this value. Regarding the Zuck medium deposit, the value obtained by DE-OPMSPs was better than or equal to all the algorithms, except the LPA

Table 7: Statistical test results between DE-OPMSPs and existing methods

	LR-MTS	SH-VND	LR-MTS-VND	GBA	BAA	GA+MF	LPA
DE-OPMSPs ***vs.***	+	+	+	+	+	+	≈

Table 8: Average mean rank by Friedman test of DE-OPMSPs and existing algorithms (smaller rank means better performance)

LR-MTS	SH-VND	LR-MTS-VND	GBA	BAA	GA+MF	LPA	DE-OPMSPs
6.55	6.2	5.2	4.25	3.0	6.3	2.95	**1.55**

and GBA. For the Marvin test instance, the proposed method was better than those of 6 algorithms, but inferior to only one. A similar performance was attained for SM2, in which DE-OPMSPs was superior to all the algorithms except LR-MTS-VND.

Statistically, the Wilcoxon test was carried out to determine if there was a statistical difference between the proposed algorithm and the other algorithms based on a whole set of test instances, i.e., 10 results. With a 5% significance level, DE-OPMSPs outperformed 5 algorithms, but no statistical difference was found between it and only the LPA.

Statistically, the Wilcoxon test was carried out to determine if there was a statistical difference between the proposed algorithm and the other algorithms based on a whole set of test instances, i.e., 10 results. With a 5% significance level, one of three symbols (+, − and ≈) was used, with +, − and ≈ meaning DE-OPMSPs was statistically superior, inferior and similar to the other algorithm, respectively. The summary given in Table 7 shows that DE-OPMSPs outperformed 5 algorithms, but no statistical difference was found between it and the LPA.

To rank all the algorithms, the Friedman test was carried out, with the summary given in Table 8 revealing that the proposed algorithm had an edge over all the other methods. Note that high values (i.e., higher than those of the worst method) were assigned to SH-VND and LR-MTS-VND for the KD and McLaughin lim test instances, as both methods were not able to solve them. The same was done for GA. Note that here, the smaller the mean rank value, the better the algorithm is.

5.3. Discussion

In this sub-section, we analyze the effect of the reduction of the number of decision variables over the planning horizon and the local search introduced in this study. To do this, different variants (var_1: DE-OPMSPs without reduction of the number of variables and local search; var_2: DE-OPMSPs with reduction of the number of variables but without local search; var_3: local search without reduction of decision variables; var_4: local search with a reduction of decision variables) are tested on the Zuck small problem, which has 20 time periods, 9400 blocks, $333,680$ constraints with $145,640$ are precedence constraints. The best result achieved by each algorithm is reported in Table 9.

The results demonstrate that the reduction of the number of variables over the time horizon (var_2) improves the standard DE's performance (var_1) by 1.49% which is equal to $12,727,317 in the total profit.

The interesting fact is that the local search (var_3), which basically depends on conventional optimization techniques discussed earlier, was not able to solve the problem when no reduction was considered, even after 24 hours on a PC with the specifications previously mentioned. This problem was clearly mitigated by carrying out the op-

Table 9: GAP% results obtained by different variants of DE-OPMSPs on Zuck small problem

var_1	var_2	var_3	var_4	DE-OPMSPs
13.13	11.64	-	1.07	0.86

timization process at each period separately, which is shown by the result achieved by var_4. Having said that, the hybridization of DE and local search with the reduction of the problem dimensionality over the time horizon is taken into account (DE-OPMSPs) led to better results. This version took a few minutes to solve this problem.

6. Conclusions and Future Work

Solving OPMSPs is a challenging research area due to the presence of a large number of decision variables and hard constraints. Although many conventional approaches and heuristics for solving them have been introduced, they still have some limitations. Over the years, EAs have shown promising performance in solving complex problems, but scant research on using them for solving OPMSPs exist.

In this paper, the first attempt to adapt DE to solve OPMSPs was introduced. The mathematical model was simplified and repeatedly optimized by the proposed algorithm over a time window that covered only one period until all periods in the time horizon ware considered. The algorithm started with real-valued solutions which were then converted to integer-based sequences with a heuristic carried out to guarantee feasibility. Then, DE operators were used to evolve the real-valued solutions which were heuristically modified to obtain feasible ones. Upon completing the optimization process for one time period, a local search procedure was carried out. The local search was based on a LP-based branch-and-cut algorithm, with the LPs being solved by the dual simplex algorithm. The algorithm continued until all the time periods were covered.

The proposed methodology was tested on 10 open-pit mine instances, both academically designed and real-life deposits. Three parameters (PS, $Prob_{prec}$ and α) were analyzed and the final version of algorithm favored setting them to values of 30, adaptive, and 20, respectively. In addition, this study has identified that the proposed algorithm had an edge over 6 techniques that recently solved the same test instances.

Overall, as the performance of evolutionary algorithms deteriorates when the number of decision variables increases, we believe that reducing the number of decision variables over the time horizon mitigates this problem, as it helps to reduce the search space, which in turn allows a better convergence rate. Also, the local search used plays a vital role in the success of the proposed algorithm. In addition, converting infeasible solutions to feasible ones helped DE to improve the quality of solutions, instead of wasting fitness evaluations to evolve infeasible ones.

Although the proposed algorithm has shown success in solving OPMSPs, some limitations were observed. The computational time consumed to repair infeasible solutions may be seen as one shortcoming. Unfortunately, this issue cannot be solely solved by using constraint-handling techniques (i.e., feasibility based approach), as such approaches cannot reach easily a feasible solution for such large-scale problems. So, to reduce the computational time of the proposed algorithm, a possible way may be dynamically using either the repair method or any of the existing constraint-handling techniques or to use a pre-process operation, possibly by using a block aggregation approach, to further reduce a problem's dimensionality. In line with this limitation, the extra process of converting discrete solutions to continuous ones (to fit the DE operators) and vice-versa adds extra time, yet, it showed its effectiveness in comparison with other EAs (i.e., GA). Therefore, using one or more EAs, which do not require extra decoding/encoding processes, along with DE within the same framework and subsequently sharing information among them efficiently might be a way to improve the algorithm's efficiency further.

Another future direction may be extending the algorithm to deal with uncertain OPMSPs. During the last decade, the evolutionary computation field has witnessed the development of many efficient evolutionary algorithms, so analysing those algorithms on these complex scheduling problems will be beneficial.

Acknowledgment

This research is supported by the Australian Research Council Discovery Project DP170102416 awarded to R. Sarker, UNSW Rector's start up grant awarded to S. Elsayed and by CONACyT project no. 2016-01-1920 and a 2018 SEP-Cinvestav grant (application no. 4) awarded to C.A. Coello Coello.

References

[1] Achterberg, T., 2009. Constraint integer programming. PhD dissertation, Zuse Institute Berlin.

[2] Achterberg, T., Wunderling, R., 2013. Mixed Integer Programming: Analyzing 12 Years of Progress. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 449–481.

[3] Andersen, E. D., Andersen, K. D., 1995. Presolving in linear programming. Mathematical Programming 71 (2), 221–245.

[4] Bartels, R. H., Golub, G. H., 1969. The simplex method of linear programming using lu decomposition. Communications of the ACM 12 (5), 266–268.

[5] Bley, A., Boland, N., Fricke, C., Froyland, G., 2010. A strengthened formulation and cutting planes for the open pit mine production scheduling problem. Computers & Operations Research 37 (9), 1641–1647.

[6] Boland, N., Dumitrescu, I., Froyland, G., Gleixner, A. M., 2009. Lp-based disaggregation approaches to solving the open pit mining production scheduling problem with block processing selectivity. Computers & Operations Research 36 (4), 1064–1089.

[7] Busnach, E., Mehrez, A., Sinuany-Stern, Z., 1985. A production problem in phosphate mining. Journal of the Operational Research Society, 285–288.

[8] Caccetta, L., Hill, S. P., 2003. An application of branch and cut to open pit mine scheduling. Journal of global optimization 27 (2), 349–365.

[9] Chicoisne, R., Espinoza, D., Goycoolea, M., Moreno, E., Rubio, E., 2012. A new algorithm for the open-pit mine production scheduling problem. Operations Research 60 (3), 517–528.

[10] Cullenbine, C., Wood, R. K., Newman, A., Aug 2011. A sliding time window heuristic for open pit mine block sequencing. Optimization Letters 5 (3), 365–377.

[11] Das, S., Suganthan, P. N., 2011. Differential evolution: a survey of the state-of-the-art. Evolutionary Computation, IEEE Transactions on 15 (1), 4–31.

[12] Davis, L., et al., 1991. Handbook of genetic algorithms. Vol. 115. Van Nostrand Reinhold New York.

[13] Denby, B., Schofield, D., 1994. Open-pit design and scheduling by use of genetic algorithms. Transactions of the Institution of Mining and Metallurgy. Section A. Mining Industry 103.

[14] Elsayed, S., Hamza, N., Sarker, R., 2016. Testing united multi-operator evolutionary algorithms-ii on single objective optimization problems. In: 2016 IEEE congress on evolutionary computation (CEC). IEEE, pp. 2966–2973.

[15] Elsayed, S., Sarker, R., Coello Coello, C. A., Jan 2019. Fuzzy rule-based design of evolutionary algorithm for optimization. IEEE Transactions on Cybernetics 49 (1), 301–314.

[16] Elsayed, S., Sarker, R., Essam, D., 2012. An improved self-adaptive differential evolution algorithm for optimization problems. IEEE Transactions on Industrial Informatics 9 (1), 89–99.

[17] Elsayed, S., Sarker, R., Ray, T., Coello, C. C., 2017. Consolidated optimization algorithm for resource-constrained project scheduling problems. Information Sciences 418-419, 346 – 362.

[18] Espinoza, D., Goycoolea, M., Moreno, E., Newman, A., 2013. Minelib: a library of open pit mining problems. Annals of Operations Research 206 (1), 93–114.

[19] Ferland, J. A., Amaya, J., Djuimo, M. S., 2007. Application of a particle swarm algorithm to the capacitated open pit mining problem. In: Autonomous robots and agents. Springer, pp. 127–133.

[20] Fu, Z., Asad, M. W. A., Topal, E., 2019. A new model for open-pit production and waste-dump scheduling. Engineering Optimization 51 (4), 718–732.

[21] Jélvez, E., Morales, N., Nancel-Penard, P., Cornillier, F., 2019. A new hybrid heuristic algorithm for the precedence constrained production scheduling problem: A mining application. Omega.

[22] Jélvez, E., Morales, N., Nancel-Penard, P., Peypouquet, J., Reyes, P., 2016. Aggregation heuristic for the open-pit block scheduling problem. European Journal of Operational Research 249 (3), 1169–1177.

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

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

[25] Kenny, A., Li, X., Ernst, A. T., Thiruvady, D., 2017. Towards solving large-scale precedence constrained production scheduling problems in mining. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, pp. 1137–1144.

[26] Khan, A., 2018. Long-term production scheduling of open pit mines using particle swarm and bat algorithms under grade uncertainty. Journal of the Southern African Institute of Mining and Metallurgy 118 (4), 361–368.

[27] Khan, A., Niemann-Delius, C., 2014. Production scheduling of open pit mines using particle swarm optimization algorithm. Advances in Operations Research 2014.

[28] Khan, A., Niemann-Delius, C., 2015. Application of Particle Swarm Optimization to the Open Pit Mine Scheduling Problem. Springer International Publishing, Cham, pp. 195–212.

[29] Khan, A., Niemann-Delius, C., 2018. A differential evolution based approach for the production scheduling of open pit mines with or without the condition of grade uncertainty. Applied Soft Computing 66, 428 – 437.

[30] Lamghari, A., Dimitrakopoulos, R., Ferland, J. A., 2015. A hybrid method based on linear programming and variable neighborhood descent for scheduling production in open-pit mines. Journal of Global Optimization 63 (3), 555–582.

[31] Liu, S. Q., Kozan, E., 2016. New graph-based algorithms to efficiently solve large scale open pit mining optimisation problems. Expert Systems with Applications 43 (Supplement C), 59 – 65.

[32] Martinez, M. A., Newman, A. M., 2011. A solution approach for optimizing long-and short-term production scheduling at lkab´s kiruna mine. European Journal of Operational Research 211 (1), 184 – 197.

[33] Myburgh, C., Deb, K., 2010. Evolutionary algorithms in large-scale open pit mine scheduling. In: Proceedings of the 12th annual conference on genetic and evolutionary computation. ACM, pp. 1155–1162.

[34] Nelis, G., Morales, N., Widzyk-Capehart, E., 2019. Comparison of different approaches to strategic open-pit mine planning under geological uncertainty. In: Proceedings of the 27th International Symposium on Mine Planning and Equipment Selection-MPES 2018. Springer, pp. 95–105.

[35] Paithankar, A., Chatterjee, S., 2019. Open pit mine production schedule optimization using a hybrid of maximum-flow and genetic algorithms. Applied Soft Computing 81, 105507.

[36] Rezakhah, M., Moreno, E., Newman, A., 2019. Practical performance of an open pit mine scheduling model considering blending and stockpiling. Computers Operations Research, 104638.

[37] Samavati, M., Essam, D., Nehring, M., Sarker, R., 2017. A methodology for the large-scale multi-period precedence-constrained knapsack problem: an application in the mining industry. International Journal of Production Economics 193 (Supplement C), 12 – 20.

[38] Schrijver, A., 1998. Theory of linear and integer programming. John Wiley & Sons.

[39] Shishvan, M. S., Sattarvand, J., 2015. Long term production planning of open pit mines by ant colony optimization. European Journal of Operational Research 240 (3), 825–836.

[40] Storn, R., Price, K., 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11 (4), 341–359.

[41] Tan, S., Ramani, R., 1992. Optimization models for scheduling ore and waste production in open pit mines. In: 23rd APCOM Symposium. pp. 781–791.

[42] Tanabe, R., Fukunaga, A., July 2014. Improving the search performance of shade using linear population size reduction. In: IEEE Congress on Evolutionary Computation. pp. 1658–1665.

[43] Zhang, J., Sanderson, A. C., 2009. Jade: adaptive differential evolution with optional external archive. IEEE Transactions on Evolutionary Computation 13 (5), 945–958.