Consolidated Optimization Algorithm for Resource-constrained Project Scheduling Problems

Saber Elsayed, Ruhul Sarker, Tapabrata Ray

School of Engineering and Information Technology,

University of New South Wales Canberra,

Australia

Carlos Coello Coello

Depto. de Computación, CINVESTAV-IPN,

Mexico

Abstract

Resource-constrained project scheduling problems (RCPSPs) represent an important class of practical problems. Over the years, many optimization algorithms for solving them have been proposed, with their performances of such algorithms evaluated using well-established test instances with various levels of complexity. While it is desirable to obtain a high-quality solution and fast rate of convergence from an optimization algorithm, no single one performs well across the entire space of instances. Furthermore, even for a given algorithm, the optimal choice of its operators and control parameters may vary from one problem to another. To deal with this issue, we present a generic framework for solving RCPSPs in which various meta-heuristics, each with multiple search operators, are self-adaptively used during the search process and more emphasis is placed on the better-performing algorithms, and their underlying search operators. To further improve the rate of convergence and introduce good-quality solutions into the population earlier, a local search approach is introduced. The experimental results clearly indicate the capability of the proposed algorithm to attain high-quality results using a small population. Compared with several state-of-the-art algorithms, the proposed one delivers the best solutions for problems with 30 and 60 activities, and is very competitive for those involving 120 activities.

Keywords: Resource-constrained project scheduling problems, evolutionary algorithms, multi-algorithm, multi-operator

1. Introduction

Resource constrained project scheduling problems (RCPSPs) represent one of the most important and challenging scheduling problems and are known to be NP-hard [10]. Such problems are at the heart of many applications, e.g., job-shop scheduling problems [17], and can be found in construction management, the production of cars, rolling ingots and assembly shop scheduling [9, 8].

The aim of an RCPSP is to find the optimal schedule of a set of activities that minimizes the total duration of the project (makespan) while satisfying some constraints. Generally, a single project consists of $D+2$ activities, i.e., $\{1, 2, .., j, ..., D+2\}$, where the first and last are dummies to be scheduled, each of which has a duration (d_j). There are different types of resources, i.e., $R = \{R_1, ..., R_k, ..., R_K\}$, with each activity requiring r_{kj} units of the k^{th} type of resource. Note that a dummy activity is one with $d_j = 0$ and $r_{kj} = 0, \forall\, k = \{1, 2, ...K\}$. In this paper, we assume that an activity in progress cannot be interrupted, and the following two kinds of constraints need to be satisfied.

1. Precedence constraint: the j^{th} activity cannot be started before the completion of all its predecessors, that is, its starting time (ST_j) is always greater than or equal to the maximum finish time (FT) of its predecessors.
2. Resource constraint: the r_k required by the j^{th} activity should be less than or equal to the available R_k at each time step.

Email addresses: {s.elsayed, r.sarker,t.ray}@unsw.edu.au (Saber Elsayed, Ruhul Sarker, Tapabrata Ray), ccoello@cs.cinvestav.mx (Carlos Coello Coello)

Generally, the mathematical model of a single-mode RCPSP can be formulated as

$$\text{minimize } FT_{D+2}$$

$$\text{subject to: } FT_j \leq FT_{j+1} - d_{j+1}, \forall\ j = \{1, 2, ..., D+1\}$$

$$\sum_{j \in A(t)} r_{kj} \leq R_k, \forall\ k = \{1, 2, ..., K\}$$

$$FT_j \geq 0, \forall\ k = \{1, 2, ..., K\} \tag{1}$$

where $A(t)$ is a set of activities scheduled at time $t \leq FT_{D+2}$.

Over the years, several approaches for solving RCPSPs, including exact, heuristics, meta-heuristics and hyper-heuristic ones, have been proposed, with exact methods only able to be used to solve small-scale problems. Although heuristics, which are rules of thumb designed to solve specific kinds of problems, perform better than exact methods for most problems, it is difficult to find a single one that performs well for a range of test instances with varying complexities [4]. Meta-heuristics, such as evolutionary algorithms (EAs) (i.e., genetic algorithms (GAs) [14] and differential evolution (DE) [36]) and swarm intelligence (SI) (i.e., ant colony optimization (ACO) [35] and particle swarm optimization (PSO) [18]), have demonstrated better performances than heuristic-based approaches [16]. Although the underlying flexibility they offer through evolving a population of solutions minimizes the chances of becoming trapped in local optima [34], due to their stochastic natures, they cannot guarantee the delivery of an optimal solution. The idea of hyper-heuristic-based approaches is to use a high-level heuristic to select a heuristic in the low-level, that is, solutions are indirectly evolved through low-level heuristics. While there is limited literature on these techniques for solving RCPSPs, our proposed method is a step in that direction.

The detailed review of the existing literature provided in Section 2 clearly reveals the following observations.

- Although some approaches were able to obtain good solutions to some problems, their quality was still far from optimal or from their corresponding lower bounds (where the optimal solution was not known) for many test instances and their rates of convergence were often slow.
- No single algorithm performed consistently well across the complete range of instances space of RCPSPs.
- Even for a single algorithm, the choice of the right set of operators involved tedious steps of parameter tuning.

To manage the above challenges, we introduce a consolidated optimization framework which has more than one optimization algorithm, each of which uses multiple operators to efficiently solve the entire spectrum of RCPSPs. The framework uses two multi-operator algorithms (MOAs). As one of our aims is to speed up the rate of convergence, we use a common population evolved by the MOAs in a sequential manner, where the probability of applying each MOA is based on its effectiveness in identifying good solutions and emphasis is placed on the best-performing operator in each MOA. This two-level optimization procedure (for selecting an algorithm and operators), which is undertaken during the overall search process is a unique feature of the proposed method.

The performance of the proposed approach is evaluated using a set of test instances with 30, 60 and 120 activities (J30, J60 and J120, respectively) is available in the project scheduling library (PSPLIB) [26, 27]. Compared with well-known state-of-the-art algorithms, the results obtained by this approach are the best in terms of solutions quality and convergence for the J30 and J60 instances and very competitive for the J120 ones. Also, the performance of the algorithm and the effects of the choices of its parameters are analyzed to offer deeper insights into their strengths and weaknesses. Different from existing techniques which suggest using a large population to solve RCPSPs [3][30], the proposed algorithm has the capability to work effectively with a small population. In fact, this can save computational time, as optimal solutions may be found in a small number of generations, that is, no need to evaluate a large number of solutions over generations.

The rest of this paper is organized as follows: an overview of related work is presented in Section 2; the proposed algorithm is described in Section 3; and the experimental results and conclusions are discussed in Sections 4 and 5, respectively.

2. Solving RCPSPs: A Brief Review

Over the years, many exact methods, heuristics, meta-heuristics and hyper-heuristic techniques for solving RCPSPs have been proposed. Of the first two approaches, integer programming-based heuristics, branch-and-bound (BB) procedures [13, 17] and single-pass (a single schedule is generated) and multi-pass (more than one schedule is repeatedly

generated) priority rule-based scheduling, are the most common. In fact, due to early unsuccessful attempts to solve RCPSPs using integer programming, several BB procedures were developed [17] which, although producing promising results, are computationally expensive for problems with a large number of activities. Generally speaking, in a priority rule-based scheduling heuristic, two components are required [25]: (1) a schedule generation process (serial or parallel); and (2) a priority rule, such as the latest finish time (LFT), most total number of successors (TS) and minimum slack time (MST) [24]. Although they are widely used because they are easy to implement and computationally cheaper than BB procedures, it is difficult to find an efficient priority rule that performs well for a broad range of RCPSPs [4].

As a consequence, meta-heuristics, such as EAs and SI, have attracted the attention of researchers and offered improvements for solving RCPSPs, with some of these algorithms discussed below.

GAs

Alcaraz and Maroto [2] proposed a GA for solving RCPSPs. In it, the activity list representation was used to encode the solutions by ensuring that any activity could appear in any location after all its predecessors, with the serial method used to generate feasible solutions based on the LFT rule. Three crossover operators were developed and tested, namely the precedence set, forward-backward (similar to the traditional one-point crossover) and two-point forward-backward (similar to the two-point crossover) crossover operators. For mutation, two operators were used to: (1) randomly move the position of each activity to another one that satisfied the precedence constraint; and (2) exchange each position with the following one. For the PSPLIB problems, although the algorithm with the forward-backward crossover and first mutation operator offered better performance than the other variants, its results were far from the best-known solutions.

Debels and Vanhoucke [16] proposed a decomposition-based GA (DBGA) that iteratively solved sub-parts of a project. In it, the population was randomly generated and a serial scheduling scheme used to convert each individual to a feasible one. Then, sub-problems were constructed and improved by a GA which used a two-point crossover with a modified version of the peak crossover [39]. The algorithm was tested on PSPLIB test instances and performed well for problems with 60 and 120 activities, and was competitive for those problems with 30. Valls et al. [40] proposed a hybrid GA (HGA) in which, a specific peak crossover operator was introduced to define the good parts of a chromosome to combine, and a double justification operator used as a local search engine. The algorithm was superior to other algorithms considered in the paper for problems with 60 and 120 activities, and was competitive for those with 30. However, it has been outperformed by many other approaches introduced in recent years.

In the GA proposed by Mendes et al. [30], chromosome representation was based on random keys, with the schedules constructed by a heuristic priority rule which was an ideal priority with an infinite capacity. The underlying GA used a uniform and one- or two-point crossover operators, and a mutation operator to introduce random solutions in the current population. This algorithm was tested on well-known benchmark problems and, although it showed competitive results, could not outperform other algorithms in the literature. A similar algorithm which was competitive to those in the literature was introduced in [21], but it suffered from a slow convergence rate. Zamani [42] proposed a GA with a magnet-based crossover (GA-MBX) that preserved up to two contiguous parts from the second parent and one contiguous part from the first parent's genotype. This type of crossover was an enhanced version of the two-point crossover. To increase the quality of solutions, a local search procedure was adopted, with the results indicating that GA-MBX could deliver competitive solutions.

DE algorithms

Cheng et al. [12] combined fuzzy clustering and a chaotic technique in a DE algorithm (FCDE), and solved two case study problems. A logistic map was used to generate chaotic sequences which were then processed by the serial generator to produce feasible schedules. DE operators evolved the population and a fuzzy c-means tracked the main movements of the population. Although this algorithm appeared to perform well in comparison with other optimization ones, this was difficult to judge, due to the limited number of test problems considered in the paper. More recently, Ali et al. [5] proposed a DE algorithm for solving RCPSPs, with its operators used to generate real-valued solutions which were then converted to integer-based ones. Also, two local search procedures combined with the scheme known as the superiority of feasible solutions [15] to handle infeasible schedules. Although this algorithm performed well, it was only tested on a small sample of benchmark problems.

Estimation of distribution algorithm (EDA)

In the EDA called HEDA introduced in [41], solutions were encoded using the extended active list (EAL) and the serial schedule generation scheme (SGS) used to generate feasible schedules, which were then updated by the EDA operators. A local search was used to improve the quality of solutions but, when evaluated on PSPLIB test instances, HEDA did not outperform the state-of-the-art algorithms considered.

SI-based approaches

Zhang et al. [44] evaluated two variants of PSO for solving RCPSPs, the first of which used priority-based representation and the second permutation-based representation with the serial method adopted to convert particles to feasible schedules. Although the results showed that the priority-based PSO representation outperformed the other, the results were inferior to those obtained by recent PSO variants [23, 10]. The algorithm was then extended slightly in [43], to generate feasible schedules using the parallel method but insufficient results were presented to compare the algorithms.

A PSO which had forward-backward improvement (FBI) and double-justification procedures for enhancing the quality of solutions was introduced in [10] but its results were far from the best-known solutions. In [23], an improved PSO for solving RCPSPs, in which particles represented using a rank-priority technique and schedules were generated using the serial method, was proposed. In addition, a double-justification local search procedure employed as a local search mechanism. The algorithm was tested on PSPLIB test instances and demonstrated better results than other PSO variants, but was significantly inferior to those of state-of-the-art algorithms.

Ziarati et al. [46] investigated the performances of three bee-type algorithms (1) the bee algorithm (BA); (2) artificial bee colony (ABC); and (3) bee swarm optimization (BSO) for solving RCPSPs. Three local search procedures were introduced in each and the results showed that BA with FBI was the best. However, all of these algorithms were inferior to many other approaches previously reported in the literature.

Although ACO was also used to solve RCPSPs [31], its performance was inferior to those of other algorithms [20, 28].

Hybrid algorithms

Fang and Wang [20] proposed a shuffled frog-leaping algorithm (SFLA), which was a combination of a memetic GA and PSO, for solving RCPSPs. In it, the virtual frog was encoded as the EAL with feasible schedules generated by the SGS. Firstly, initial solutions were generated by the regret-based sampling method and priority rule, and then evolved by adopting a crossover operator, with a combined local search, i.e., a permutation-based local search (PBLS) and FBI. The algorithm was tested on PSPLIB test problems, with the results showing competitive performances on problems with 120 and 60 activities, but a poor performance for those problems with 30.

Agarwal et al. [1] proposed a hybrid framework combining a GA and neural network (NN) (GANN), in which the GA operators were used to search for good solutions and the NN for a local search. However, the algorithm did not perform well. Chen et al. [11] proposed a hybrid algorithm for tackling RCPSPs, in which ACO and a scatter search (SS) were used in an iterative manner. Firstly, ACO generated new solutions which were then passed on to the SS algorithm for improvement and then ACO used the improved solutions to update the pheromone set. Also, a local search procedure was applied to enhance the quality of solutions. Although the results were not better than those reported in the literature for problems with 30 and 60 activities, they were good for those problems with 120.

Hyper-heuristic-based algorithms

A few hyper-heuristic-based approaches for solving RCPSPs have been proposed; for example, Koulinas et al. [28] developed a PSO-based hyper-heuristic algorithm in which the solutions were represented using random keys and schedules constructed by the SGS using the priorities of the activities. These priorities were updated by one of 8 low-level heuristics. Also, each particle was a vector of 8 integer numbers, each representing a low-level heuristic, with particles indicating the order in which the low-level heuristics were applied. In addition, a local search procedure was applied to every generated solution. The algorithm showed competitive performance, but it was noted that its convergence was not very good. Anagnostopoulos and Koulinas [6] proposed a genetic hyper-heuristic algorithm. The lower level had 8 heuristics, 6 of which included random selection-based heuristics and the remaining two selected activities based on the TS criterion while, in the upper level, the random linear bias selection was adopted to choose the best heuristic. The algorithm showed encouraging results but the paper did not provide a proper comparison with other existing approaches. Following the same algorithmic framework, the same authors proposed a similar algorithm [7] by replacing the GA with a greedy randomized adaptive search procedure (GRASP)-based hyper-heuristic.

Other techniques

In [32], an approach for automatically selecting algorithms for multi-mode RCPSPs (MRCPSPs) was proposed. Firstly, to find the features of a problem that are the most suitable for an algorithm, machine-learning techniques were used and then redundant and uninformative features eliminated. Finally, a model was built for each algorithm to map the feature space onto the running time. Two algorithms were incorporated in the technique to validate its accuracy and, although encouraging results were reported, in our opinion, this approach is computationally expensive in terms of model building. Also, it might not work for solving problems that have features different from those considered in the training phase. Therefore, one of our aims in this paper is to automatically place emphasis on the best-performing algorithm during the course of the optimization process.

Algorithm 1 General framework of COA

1: Define PS, $cy \leftarrow 0$, $prob_i \leftarrow 1$, $g \leftarrow 1$, and all other parameters required (Section 4).
2: Generate random individuals (X).
3: Convert X to feasible schedules (X^s), and obtain corresponding continuous vectors X^{cont}.
4: Apply local search on X^s and updat $X = X^s$ and X^{cont}.
5: **while** $cfe < FFE_{max}$ **do**
6: $cy \leftarrow cy + 1$.
7: **if** $cy = CS$ **then**
8: Measure the improvement of each Alg_i (Section 3.1).
9: Update $prob_i$.
10: **end if**
11: **if** $cy = 2 \times CS$ **then**
12: $prob_i \leftarrow 1$.
13: $cy \leftarrow 0$; .
14: **end if**
15: **for** $i = 1 : noAlg$ **do**
16: Generate $rand_i \in [0, 1]$, with at least one satisfies $rand_i \in [0, 1] \leq prob_i$.
17: **if** $rand_i \leq prob_i$ **then**
18: Generate new offspring using Alg_i and update probability of applying each operator (Section 3.4).
19: Apply local search.
20: Update cfe.
21: **end if**
22: **end for**
23: $g \leftarrow g + 1$, and go to step 5.
24: **end while**

3. Consolidated Optimization Algorithm for RCPSPs

In this section a consolidated optimization algorithm (COA) is discussed and its steps given in Algorithm 1.

Firstly, an initial integer population (X) of size PS is randomly generated $\left(X = \left\{\overrightarrow{X}_1, \overrightarrow{X}_2, ..., \overrightarrow{X}_{PS}\right\}\right)$. Then, X is converted to feasible schedules (X^s) with their corresponding continuous vectors (X^{cont}) calculated (see Section 3.2). The makespan of each $\overrightarrow{X}^s_z \forall\, z = \{1, 2, ..PS\}$, which represents the quality of a solution, is calculated and then a local search is applied to improve each $\overrightarrow{X}^s_z$ and its $\overrightarrow{X}^{cont}_z$ is also generated, as discussed in Section 3.3.

Then, the population is then evolved by one or two multi-operator EAs (MOEAs). Initially, the probability of each algorithm being applied is set to 1, i.e., $prob_i = 1 \forall\, i = \{1, 2\}$. Then, in each generation, two random numbers are generated, i.e., $rand_i \in [0, 1]\ \forall\, i = \{1, 2\}$, and if $rand_i \leq prob_i$, new individuals are generated using Alg_i, where Alg_1 is a multi-operator GA (MOGA) and Alg_2 a multi-operator DE (MODE).

If MOGA is applied, integer-based crossover and mutation operators are used to generate new integer-based offspring (X^{new}) and then new schedules ($X^{s,new}$) and continuous-based vectors ($X^{cont,new}$) are generated (see Section 3.2). On the other hand, if MODE is adopted, using X^{cont}, firstly, new $X^{cont,new}$ are generated and then their integer-based solutions (X^{new}) calculated, with new feasible $X^{s,new}$ and $X^{cont,new}$ are subsequently produced. For both MOGA and MODE, a local search process is applied to every new offspring and then the selection procedure defines the new solutions that should survive to the next generation. Once this is done, the improvement achieved by each operator is measured to place more emphasis on the best-performing one in each MOEA. This process is repeated for a cycle (CS) and, when it is finished, every $prob_i$ is updated based on its performance during the first cycle, as discussed in Section 3.1.

For each generation in the 2^{nd} cycle, 2 random numbers are generated ($rand$), with at least one having to be less than its corresponding $prob_i$ to make sure that at least one algorithm is applied in each generation. Similar to the first cycle, if $rand_i \leq prob_i$, then Alg_i is used to update the population $\forall\, i = \{1, 2\}$ and, at the end of this cycle, each $prob_i$ is set to 1, i.e., returned to its value in the 1^{st} cycle. The reason for this is that the search capabilities of optimization algorithms may vary during the optimization process, i.e., one may be good in the early stages of but perform poorly in later generations. COA continues until a stopping criterion is met.

In the following subsections, each component of the proposed COA is discussed in detail.

3.1. Updating $prob_i$

The quality of solutions obtained by each MOEA is used to place emphasis on the best-performing one. To do this, in each generation, the best fitness value before ($f_{best,old}$) and after ($f_{best,new}$) applying each MOEA is recorded. Next,

the improvement rate ($I_{cy,i} \forall i = \{1, 2\}$), where cy is a generation counter, is calculated as

$$I_{cy,i} = \frac{f_{best,old} - f_{best,new}}{f_{best,old}}, \tag{2}$$

At the end of the 1^{st} cycle, i.e., $cy = CS$, each $prob_i$, $\forall i = \{1, 2\}$, is updated as

$$prob_i = \max\left(0.1, \min\left(0.9, \frac{\sum_{cy=1}^{CS} I_{i,cy}}{\sum_{cy=1}^{CS} I_{1,cy} + \sum_{cy=1}^{CS} I_{2,cy}}\right)\right) \tag{3}$$

Also, in the case that no improvement in the makespan is attained either algorithm, both *prob*s are set to a value of 0.5. Note also that, as one MOEA may perform well in different stages of the evolutionary process, and poorly perform in others, a minimum value of 0.1 is considered [19].

3.2. Representation and feasible schedules

As previously discussed, two MOEAs are used. As the first deals with integer-based solutions and the other with real-valued solutions, in this paper, it is essential to consider two representations.

Generally, the initial population (X) is generated in an activity list form, i.e., PS permutation vectors, each of which $\left(\overrightarrow{X}_z\right)$ consists of $D + 2$ values, where the first and last activities are dummies. As these random solutions may not satisfy the precedence and/or resource constraints, it is important to convert them to feasible ones. To do this, the SGS is used to decode each $\overrightarrow{X}_z$, $\forall z = \{1, 2, .., PS\}$, to obtain its corresponding schedule $\left(\overrightarrow{X}^s_z\right)$ by selecting the activities according to their order in $\overrightarrow{X}_z$ and scheduling them at their earliest starting times. Subsequently, the corresponding continuous-based solution $\left(\overrightarrow{X}^{cont}_z\right)$ is generated, so that each value $\overrightarrow{X}^{cont}_{z,j}$, $\forall j = \{2, ..., D + 1\}$, is the location of the j^{th} activity in $\overrightarrow{X}^s_z$ plus a random value $\in [0, 1]$, while the first ($d = 1$) and last ($d = D + 2$) values are $rand \in [0, 1]$ and $D + (rand \in [0, 1])$, respectively. To clarify this, Figure 1 shows an example of converting a feasible schedule to a continuous-based solution; for instance, as activity 3 is in the 2^{nd} position in $\overrightarrow{X}^s_z$, the 3^{rd} value in $\overrightarrow{X}^{cont}_z$ takes a value of $2 + rand_3$, where $rand_3 \in [0, 1]$. Similarly, activity 5 is in the 3^{rd} position in $\overrightarrow{X}^s_z$, the 5^{th} value in $\overrightarrow{X}^{cont}_z$ is set to a value of $3 + rand_5$, where $rand_5 \in [0, 1]$. The same is done for the remaining activities which means that the search space when using MODE is $[0, D + 3]$.

$\overrightarrow{X}^s_z$	1	3	5	2	4	6	7

$\Downarrow$

$\overrightarrow{X}^{cont}_z$	$rand_1$	$4 + rand_2$	$2 + rand_3$	$5 + rand_4$	$3 + rand_5$	$6 + rand_6$	$7 + rand_7$

Figure 1: Conversion feasible schedule to a continuous-based solution with 7 (5 + 2 dummies) activities (where $rand_j \in [0, 1], \forall j = 1, 2, ..., 7$)

For MODE, as we are dealing with real-valued vectors, $\overrightarrow{X}^{cont}_z$ is converted to $\overrightarrow{X}_z$, followed by the same steps mentioned above. To do this, firstly, we sort the values in $\overrightarrow{X}^{cont}_z$ in an ascending order and then $\overrightarrow{X}_z$ is considered the place of each activity in the rankings (Figure 2).

act	1	2	3	4	5	6	7
$\overrightarrow{X}^{cont}_z$	0.23	4.35	2.8	5.45	3.95	6.11	7.78

$\Downarrow$sort

act	1	3	5	2	4	6	7
$\overrightarrow{X}^{cont}_z$	0.23	2.8	3.95	4.35	5.45	6.11	7.78

$\Downarrow$final

$\overrightarrow{X}_z$	1	3	5	2	4	6	7

Figure 2: Conversion of real-valued solution to integer-based one

3.3. Local search

In this research, motivated by the work presented in [29], the local search starts by sorting all the activities in a descending order based on their finishing times. Then, without violating the resource and successor constraints, each activity is shifted to the right as far as possible. After this step, all the activities are sorted again based on their starting times and a feasible schedule is generated $\left(\overrightarrow{X}^{new}_{z,LS}, \forall\, z = \{1, 2, ..PS\}\right)$, as discussed in Section 3.2, i.e., shifting them to the left.

To maintain diversity within the current population, if $\overrightarrow{X}^{new}_{LS}$ is not in the memory (M_{LS}), its objective value $\left(f\left(\overrightarrow{X}^{new}_{LS}\right)\right)$ is calculated and if $f\left(\overrightarrow{X}^{new}_{LS}\right) < f\left(\overrightarrow{X}^{new}\right)$, $\overrightarrow{X}^{new}_{LS}$ replaces $\overrightarrow{X}^{new}$. On the other hand, if the solution is in M_{LS}, $\overrightarrow{X}^{new}$ is retained. It is worth highlighting the following two points about this step.

1. A solution is considered redundant, if all the starting times of its activities match those of at least one schedule in M_{LS}.
2. As retaining all the solutions generated in the previous generations increases the computational time, the maximum size of the memory is set to $2PS$ (the size of the old population plus the new one). In each generation, and before applying every MOEA, M_{LS} is cleared, and then the current solutions, $\overrightarrow{X}$, are inserted into it, i.e., the memory size is now PS. Then, if $\overrightarrow{X}^{new}_{z,LS}$ satisfies the condition previously mentioned, it is inserted into it.

3.4. Optimization algorithms

Although the framework proposed in this paper is general and can be applied to any diverse group of algorithms, two MOEAs are considered (1) MOGA which combines two GA-based crossover operators; and (2) MODE which uses two DE-based mutation operators.

3.4.1. MOGA

Firstly, of the set of solutions passed to MOGA, a tournament selection is undertaken to create a mating pool of solutions. Then, in each generation, the number of individuals that can be generated by each crossover is determined as

$$n_1 = \min(PS - 1, (\max(1, \text{count}\,(rand \in [0,1]_{1:PS} \leq P_{GA,1}))) \tag{4}$$

$$n_2 = PS - n_1 \tag{5}$$

where n_1 individuals are generated by a two-point crossover and n_2 by a uniform one, with the initial value of $P_{GA,1}$ set to 0.5. Note that a minimum value of 1 is assigned to each n_i, $i = \{1, 2\}$ and both crossover operators generate integer-based solutions (X^{new}). In a two-point crossover, from two parents P^1 and P^2, two offspring are generated as follows

1. Select two points $\in [1, D + 2]$, i.e., divide both P^1 and P^2 into three parts.
2. Copy the 1$^{\text{st}}$ and 3$^{\text{rd}}$ parts from P^1 into their corresponding indices in the first child while the remaining values are taken from P^2, with the constraint that there are no redundant activities.
3. To generate the 2^{nd} offspring, the 1^{st} and 3^{rd} parts of the variables are taken from P^2, and the remaining ones from P^1, with the constraint that there are no redundant activities.

In the uniform crossover, one child is produced from two parents P^1 and P^2 as follows

1. For every gene, i.e., $j = 1, 2, .., D$, a random number $(rand_j) \in [0, 1]$ is generated. Then, if $rand_j \leq 0.5$, the j^{th} gene from P^1 is copied.
2. Then, to avoid having redundant activities in the new child, the remaining activities, which are not taken from P^1, are copied from P^2 based on the first-come, first-served (FCFS) rule.

Then, a left-shift mutation operator is applied to each offspring generated. In it, for each gene in each chromosome $\left(X^{new}_{z,j}\right)$, a random number$(rand_j) \in [0, 1]$ is generated, and if it is less than the mutation rate(MR), $X^{new}_{z,j}$ is randomly shifted to a position to its left side, i.e., between $[1, j - 1]$.

Subsequently, X^{new} is converted to feasible schedules, $X^{s,new}$, with their corresponding $X^{cont,new}$ determined, as discussed in Section 3.2. Then, the objective values for all the new offspring are calculated, and a local search is applied to every $\overrightarrow{X}^{new}_{z}$. As elitism is important in a GA, the best solution from the previous generation is combined with the current offspring so that the best PS, among $PS + 1$, solutions survive to the next generation.

Then, the average improvement in the objective values of all the offspring generated by each GA is calculated as

$$I_{GA_\kappa} = \frac{\sum_{z=1}^{PS} max\left(0, f_{new_z} - f_{old_z}\right)}{PS}, \quad \forall \overrightarrow{X}_z \text{ updated by } \mathrm{GA}_\kappa \text{ and } \kappa = 1,2 \quad (6)$$

where f_{new} and f_{old} are the new and old fitness values, respectively.

Then, each probability is updated as

$$P_{GA,\kappa} = \max\left(0.1, \min\left(0.9, \frac{I_{GA_\kappa}}{I_{GA_1} + I_{GA_2}}\right)\right), \quad \forall\, \kappa = 1,2 \quad (7)$$

If $\sum_{\kappa=}^{2} I_{GA_\kappa} = 0$, each $P_{GA,\kappa}$ is set to a value of 0.5.

3.4.2. MODE

This algorithm starts with the set of solutions previously generated by the MOGA, or using its own steps if the MOGA is not applied during the current generation. Similar to MOGA, the number of individuals that can be generated by each DE variant in each generation is determined, as in equations 4 and 5, so that n_1 individuals are generated by DE_1 and n_2 by DE_2, respectively, as

(1) DE_1: current-to-rand/bin with archive

$$u_{z,j} = \begin{cases} X_{z,j}^{cont} + F_z\left(X_{r_1,j}^{cont} - X_{z,j}^{cont} + X_{r_2,j}^{cont} - \widetilde{X}_{r_3,j}^{cont}\right) \\ \qquad\qquad if\,(rand \leq cr_z \text{ or } j = j_{rand}) \\ X_{z,j}^{cont} \qquad\qquad \text{otherwise} \end{cases} \quad (8)$$

(2) DE_2: current-to-rand/bin without archive

$$u_{z,j} = \begin{cases} X_{z,j}^{cont} + F_z\left(X_{r_1,j}^{cont} - X_{z,j}^{cont} + X_{r_2,j}^{cont} - X_{r_4,j}^{cont}\right) \\ \qquad\qquad if\,(rand \leq cr_z \text{ or } j = j_{rand}) \\ X_{z,j}^{cont} \qquad\qquad \text{otherwise} \end{cases} \quad (9)$$

where $r_1 \neq r_2 \neq r_3 \neq z$ are random integer numbers, with $\overrightarrow{X}_{r_1}^{cont}$, $\overrightarrow{X}_{r_2}^{cont}$ and $\overrightarrow{X}_{r_4}^{cont}$ randomly selected from X^{cont} [45], while $\widetilde{X}_{r_3,j}^{cont}$ is chosen from the union of the entire X^{cont} and the archive (AR). Initially, the archive is empty, and then parent vectors which failed in the selection process are added to it. Once its size exceeds a threshold (of size PS [45]), randomly selected elements are deleted to make space for the newly inserted ones.

As both DE variants deal with continuous-based solutions, the new solutions are converted to integer-based ones (X^{new}), as discussed in Section 3.2. Subsequently, X^{new} is converted to feasible schedules, $X^{s,new}$, and their corresponding $X^{cont,new}$ are updated (Section 3.2). Then, the objective values for all the solutions are calculated and for every $\overrightarrow{X}_z^{new}$, $\forall\, \{z = 1, 2, .., PS\}$, a local search is carried out to produce$\overrightarrow{X}_{z,LS}^{new}$, which replaces $\overrightarrow{X}_z^{new}$, if its objective value is better.

After that, the average improvements (I_{DE_1}, I_{DE_2}) in the objective values are calculated in a similar way as that described in equation (6) and each DE probability is updated as

$$P_{DE,\kappa} = \max\left(0.1, \min\left(0.9, \frac{I_{GA_\kappa}}{I_{GA_1} + I_{GA_2}}\right)\right), \quad \forall\, \kappa = 1,2 \quad (10)$$

If $\sum_{\kappa=}^{2} I_{DE_\kappa} = 0$, then each $P_{DE,\kappa}$ is set to a value of 0.5.

Adaptation of F and Cr

In this paper, the mechanism proposed in [38], which is considered an improvement of JADE [45], is adopted and works as follows:

- Initially, H entries for both parameters (M_{Cr}, M_F) are initialized, with all the values are set to 0.5.

- Each individual $\overrightarrow{x}_z$ is associated with its own Cr_z and F_z as

$$Cr_z = \text{randni}(M_{Cr,r_z}, 0.1) \tag{11}$$

$$F_z = \text{randci}(M_{F,r_z}, 0.1) \tag{12}$$

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 variance of 0.1.

- At the end of each generation, the Cr_z and F_z used by the successful individuals are recorded in S_{Cr} and S_F, respectively, and then the contents of memory are updated as

$$M_{Cr,d} = mean_{WA}\left(S_{Cr}\right) \; if \; S_{Cr} \neq \text{null} \tag{13}$$

$$M_{F,d} = mean_{WL}\left(S_F\right) \; if \; S_F \neq \text{null} \tag{14}$$

where $1 \leq d \leq H$ is a position in the memory to be updated. It is initialized to 1 and then incremented whenever a new element is inserted into the history, 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 [1]

$$\text{mean}_{WA}(S_{Cr}) = \sum_{\gamma=1}^{|S_{Cr}|} w_\gamma S_{cr,\gamma} \tag{15}$$

$$\text{mean}_{WL}(S_F) = \frac{\sum_{\gamma=1}^{|S_F|} w_\gamma S_{F,\gamma}^2}{\sum_{\gamma=1}^{|S_F|} w_\gamma S_{F,\gamma}} \tag{16}$$

where

$$w_\gamma = \frac{\Delta f_\gamma}{\sum_{\gamma=1}^{|S_{cr}|} \Delta f_\gamma} \tag{17}$$

and $\Delta f_\gamma = |f_{\gamma,old} - f_{\gamma,new}|$.

4. Experimental Results

This section presents, discusses and analyzes the computational results obtained by COA on the PSPLIB instances [26, 27] with 30, 60 and 120 activities (J30, J60 and J120, respectively). The J30, J60 and J120 sets contain 48, 48 and 60 test instances, respectively, with 10 problems in each instance (this makes a total of 480, 480 and 600 problems, respectively) which were generally generated by changing three parameters:

(1) the network complexity (NC) which defines the average number of predecessors per activity.

(2) the resource factor (RF) which determines the average percentages of different resource types for which each activity has a nonzero-demand.

(3) the resource strength (RS) which defines the degree of scarceness of the resources.

For the J30 and J60 instances, the parameter values were: $NC \in \{1.5, 1.8, 2.1\}$, $RF \in \{0.25, 0.5, 0.75, 1\}$ and $RS \in \{0.2, 0.5, 0.7, 1\}$ which were different from those for the J120 ones of $NC \in \{1.5, 1.8, 2.1\}$, $RF \in \{0.25, 0.5, 0.75, 1\}$ and $RS \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ [31]. It is worth mentioning that if RF had a value of 1, this would mean that every activity needed all the resources to be completed, while the opposite exists when $RF = 0$. Also, for $RS = 0$, the capacity of each resource cannot exceed the maximum demand of all activities, while 1 means that the capacity of each resource is equal to the demand imposed by a project performed based on the earliest start time schedule [31, 22].

The computational results were recorded at three different levels, i.e., 1000, 5000 and 50000 fitness evaluations with the COA's parameters $PS = 10$ and $CS = 25$ (the only new parameter introduced). For MOGA, the mutation rate (Mr_{GA}) was set to a value of 0.05 at the beginning of the evolutionary process and then linearly reduced to 0.001 (i.e., $= 0.05 - 0.049 \times \frac{cfe}{50000}$, where cfe is the current number of fitness evaluations), the crossover rate (Cr_{GA}) 1, the tournament size 2, and elitism size 1 while, for MODE, H was set to the number of activities [37] with 15 runs carried out.

[1] $|S_{Cr}|$ is the number of successful Cr recorded in S_{Cr}, with $|S_{Cr}|$=$|S_F|$

Table 1: Average deviations obtained by COA for J30, J60 and J120 for 1000, 5000 and 50000 fitness evaluations

	Problems	Schedules		
		1000	5000	50000
$\overline{Dev}$	J30	0.03	0.01	0.00
	J60	11.11	10.79	10.60
	J120	34.04	32.90	31.198

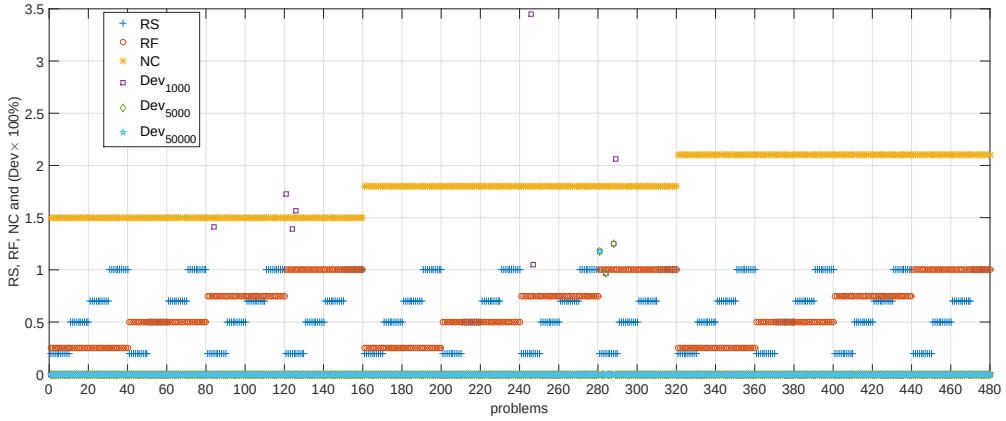


Figure 3: Deviation from lower bound for each problem with 30 activities for 1000, 5000 and 50000 schedules (deviation values in %)

The deviation (Dev) from the lower bound (LB) was calculated for each problem (p) as

$$Dev_p = \frac{F_{best,p} - LB_p}{LB_p}, \forall p = 1, 2, ..., P \tag{18}$$

where P is the maximum number of problems, i.e., 480, 480 and 600 for J30, J60 and J120, respectively, and $F_{best,p}$ the best fitness value achieved for the p^{th} problem. Subsequently, the average deviation $\left(\overline{Dev}\right)$ was calculated as

$$\overline{Dev}_D = \frac{\sum_{p=1}^{P} Dev_p}{P} \times 100\%, \forall\, D = \{30, 60, 120\} \tag{19}$$

Note that all the experiments were run on a PC with a Core(TM) i7-3770 CPU @ 3.40GHz, 16 GB RAM and Windows 7 using MATLAB 8.5.0 (R2015a) with some of its parts converted to a C++ code by the MATLAB coder. Note that MATLAB was run in an ordinary serial mode.

From the results obtained (Table 1 and Figure 3), it was found that COA was fast to obtain the optimal solutions for 470, 477 and 479 of the 480 J30 problems, in 1000, 5000 and 50000 schedules, respectively. The only problem for which COA could not obtain the optimum value was 281 (instance 29 problem 1), in which the best makespan obtained was 86, whereas the optimal one would be 85, with only 1000 schedules, but did subsequently improve. Another observation was that those problems for which COA could not achieve optimality with 1000 schedules had RS values of 0.2 (the smallest in the range used to generate the test instances).

For the J60 instances, as their optimal solutions were not known, and their LBs, which are not often feasible, were used to calculate the $\overline{Dev}$, COA's deviations were expected to be higher than those of the J30 ones. However, the results were quite consistent with those of J30, in which deviations were high for those problems with small values of RS and high RF, see Figure 4. Another observation was that the algorithm was fast to obtain high-quality solutions with 1000 fitness evaluations, and then slightly improved until reaching 50000. Generally speaking, COA still coped well with the J60 instances.

As smaller values of RS were used to generate the J120 problems, this became challenging. As depicted in Figure 5, the deviations were high for small values of RS, and improved with increases in them.

In summary, it was found that both RS and RF greatly affected a problem's complexity while NC did not demonstrate any significant influence.

4.1. Parametric analysis

In this subsection, three sets of experiments designed for the J60 problems to analyze the effects of: (1) CS, (2) PS, and (3) Mr_{GA} are described.

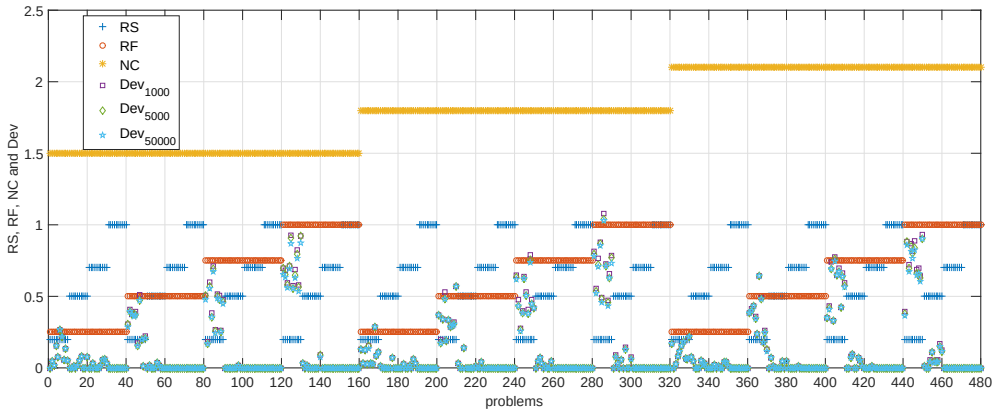


Figure 4: Deviations from lower bounds for each problem with 60 activities in 1000, 5000 and 50000 schedules

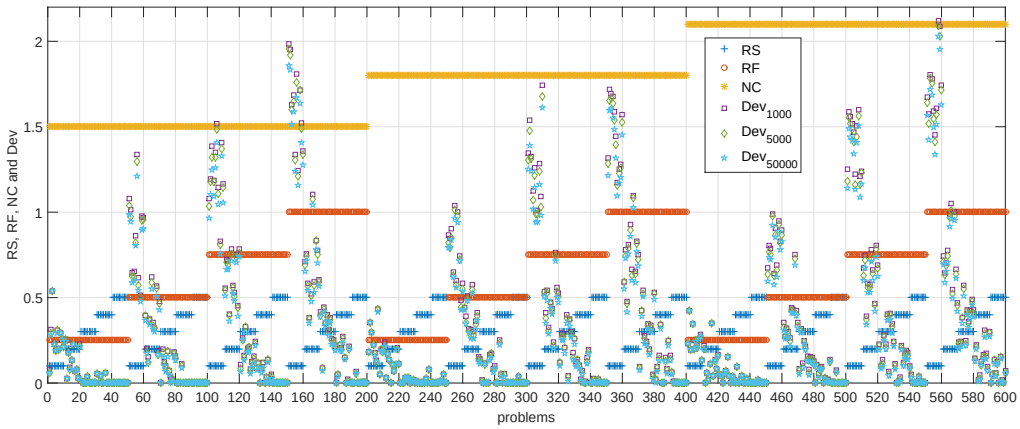


Figure 5: Deviations from lower bounds for each problem with 120 activities in 1000, 5000 and 50000 schedules

Table 2: Average deviation values obtained by COA for J60 instances in 1000, 5000 and 50000 fitness evaluations with different values of CS

	CS	Schedules			Time (seconds)
		1000	5000	50000	
$\overline{Dev}$	10	11.12	10.83	10.61	8.92
	25	**11.11**	10.79	10.60	9.27
	50	11.13	**10.77**	**10.58**	**8.52**
	100	11.13	**10.77**	10.59	8.85

Table 3: Average deviation values obtained by COA for J60 instances in 1000, 5000 and 50000 fitness evaluations with different values of PS

	PS	Schedules			Time (seconds)
		1000	5000	50000	
$\overline{Dev}$	5	11.15	10.85	10.66	10.02
	10	**11.13**	**10.77**	10.58	**8.52**
	25	11.32	10.84	10.59	8.61
	50	11.62	10.99	**10.56**	9.46

CS

The aim of the first experiment was to analyze the effect of the new parameter introduced by COA, CS, by changing it to 10, 25, 50 and 100 while Mr_{GA} and PS were set to their initial settings previously mentioned. The computational results, shown in Table 2, demonstrate that CS did not have a significant impact on COA in 1000 fitness evaluations, while its values in 5000 and 50000 improved a little until reaching 50 generations and then slightly deteriorating. Therefore, COA with a CS value of 50 was considered the best.

Also, for each variant, the average computational times taken to solve all the test problems were recorded, if one of the following two criteria was met: (1) the maximum number of fitness evaluations was reached, or (2) the best known solution was found. The results reported in Table 2 demonstrate that there was a little difference in the computational times of all the variants.

PS

In this set of experiments, four variants of COA were run by varying PS as 5, 10, 25, and 50 with Mr_{GA} set as adaptive, as discussed in Section 4, and $CS = 50$ (the best value previously found). From the computational results shown in Table 3, it is clear that increases in PS negatively affected the quality of solutions in 1000 and 5000 fitness evaluations although, interestingly, the variant with $PS = 50$ was the best in 50000 schedules. Considering the computational times, COA with $PS = 10$ was the best and generally, as one of our aims was to increase the algorithm's convergence, this size was preferred. It is interesting to conclude that we found that using a small rather than large PS was more beneficial for solving RCPSPsin contrast to the opinion of the authors in [3][30].

$\mathbf{Mr_{GA}}$

To analyze the effect of Mr_{GA}, COA was run with 8 different settings, i.e., 0.001, 0.01, 0.05, 0.1, adaptive from 0.05 to 0.001 (adaptive$_{0.05-0.01}$), adaptive$_{0.1-0.01}$, adaptive$_{0.05-0.001}$ and adaptive$_{0.1-0.001}$, while PS and CS were set to 10 and 50, respectively, as previously discussed. The results presented in Table 4 show that, although small values of Mr_{GA} could lead to high convergence rates during the first 1000 fitness evaluations, COA became stuck in local solutions later. Overall, COA performed better with adaptive reductions of Mr_{GA}, with adaptive$_{0.05-0.001}$ considered the best.

Table 4: Average deviation values obtained by COA for J60 in 1000, 5000 and 50000 fitness evaluations with different values of Mr_{GA}

	Mr_{GA}	Schedules			Time (seconds)
		1000	5000	50000	
$\overline{Dev}$	0.001	11.22	11.11	10.918	16.48
	0.01	**11.09**	10.93	10.745	11.63
	0.05	11.16	10.80	10.591	8.60
	0.1	11.23	10.92	10.631	8.68
	Adaptive$_{0.05-0.01}$	11.14	10.79	10.589	9.22
	Adaptive$_{0.1-0.01}$	11.24	10.90	10.583	8.59
	Adaptive$_{0.05-0.001}$	11.13	**10.77**	**10.581**	**8.52**
	Adaptive$_{0.1-0.001}$	11.23	10.89	10.591	8.82

Table 5: Average deviation values obtained by COA for J30, J60 and J120 in 1000, 5000 and 50000 fitness evaluations

	Problems	Schedules			Time (seconds)
		1000	5000	50000	
$\overline{Dev}$	J30	0.04	0.00	0.00	0.16
	J60	11.13	10.77	10.581	8.52
	J120	34.04	32.90	31.216	134.54

Table 6: Average deviation values obtained by COA for J120 with adaptive Mr_{GA} from 0.025 to 0.001 ($\text{COAs}_{\text{J120},Adaptive_{0.025-0.001}}$)

Problems	Schedules			Time (seconds)
	1000	5000	50000	
$\mathbf{COAs}_{\mathbf{J}120,Adaptive_{0.025-0.001}}$	33.68	32.35	31.23	139.54

4.2. Final COA

Based on the above analysis, COA with $PS = 10$, $CS = 50$ and Mr_{GA} adaptively reduced from 0.05 to 0.001 was used to solve the J30 , J60 and J120 problems, with the results shown in Table 5 being slightly different from those obtained from COA variant presented in Table 1.

A further variant was carried out to increase the convergence rate of COA when solving J120. This was done by reducing Mr_{GA} from 0.025 to 0.001 $\left(\text{COAs}_{\text{J}120,Adaptive_{0.025-0.001}}\right)$ with the results showing that it improved for1000 and 5000 schedules (Table 6). However, as this setting did not work well for the other dimensions, this variant was considered an option for only J120.

4.3. Discussion

In this section, we try to determine which algorithm and operator were preferred for each problem.

To do this, we recorded two sets of data. The first measured how many times each algorithm, at the end of the first cycle, had a higher probability of being used, as shown in the first subfigures in Figures 6 to 8, where blue indicates that no MOEA was used, i.e., the optimal solution was found before applying any MOEA, red both MOEAs had the same probability, black MOGA had a higher probability than MODE and green the opposite. The second data recorded measured the average probability of each operator over the entire optimization process.

It can be seen in Figure 6a that for the majority of J30 test problems with high RC values, the optimal solution was found before using any MOEA while for the other, no single MOEA was the best for all types of problems; for instance, MOGA was preferred for a small value of RC and $NC = 1.5$, and MODE for $NC = 1.8$ and small RC values. It is also noted that for some test problems, three cases occurred, i.e., one MOEA had a higher, equal or less probability than the other one which meant that one algorithm might be good in some generations of the evolutionary process but poor in others which, in fact, was consistent with the motivation of this paper.

Regarding the operators preferred for each MOEA, overall, it was difficult to conclude which GA operator was the best, as both MOGA ones had the same probability when $NC \leq 1.8$ and RC was high. For MODE, DE_1 had a higher probability than DE_2, for many problems (Figure 6a).

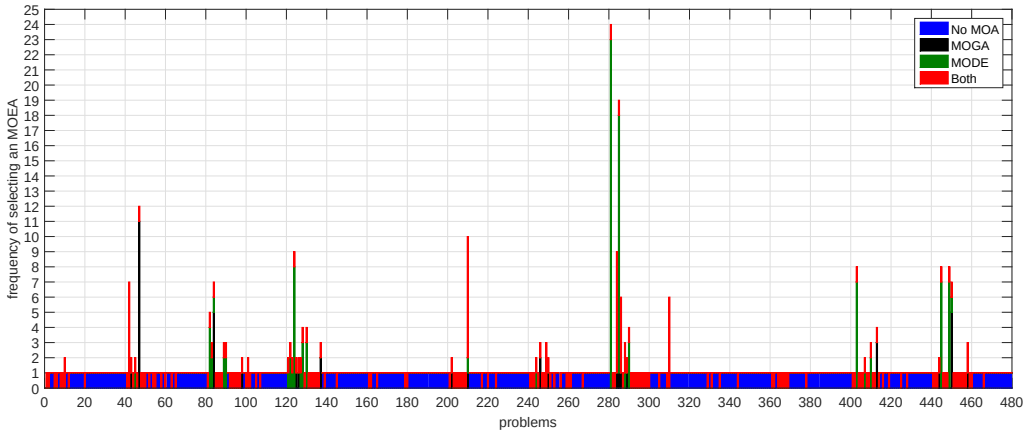
Figure 7a shows that, for J60, MODE obtained to get a higher probability than MOGA for many problems, such as 81 to 90, 128 to 131, and 244 to 246, which all had small values of RC. However, this observation was not general, as, for some problems in the same category, such as 123 to 127, 243, 247 and 441 to 444, MOGA was preferred. Regarding the probability of each operator, variations occurred when $RC \leq 0.5$ and $NC \leq 1.8$, and $RC \leq 0.7$ with $NC = 2.1$, with COA preferring GA_1 more to GA_2, although GA_2 had a higher probability when NC was high and RC small. Also, DE_1 and DE_2 had close possibilities of running, especially when $RC = 1.0$. DE_1 was slightly preferred when $RC = 0.5$ and NC was small, but the opposite happened when NC was high.

For J120, as depicted in Figure 8, it was found that MODE had a higher probability than MOGA for most of the test problems. Regarding the probability of each operator, DE_1 and DE_2 had close possibilities of running. For MOGA, it was clear that GA_1 had a higher probability than GA_2 for most problems, but GA_2 was preferred for some with $RC = 0.5$ and high NC.

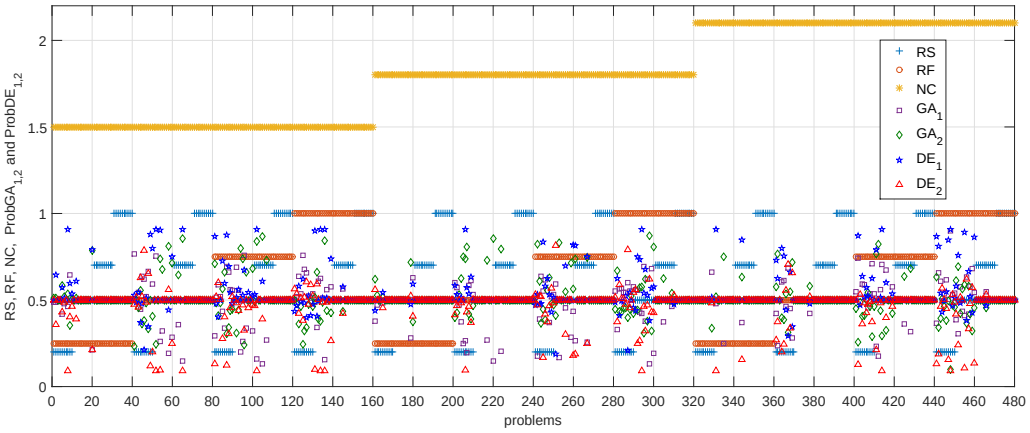
4.4. Comparison with state-of-the-art algorithms

COA was compared with 11 algorithms: (1) DBGA [16], (2) HGA [40], (3) GA [30], (4) ACO+SS [11], (5) PSO [10], (6) BA [46], (7) GANN [1], (8) HEDA [41], (9) GA-MBX [42], (10) SFLA [20], and (11) a PSO-based hyper-heuristic (PSO-HH) [28]. Note that the results of those algorithms were taken from their original papers. All these algorithms were discussed in Section 2 and comparison summaries presented in Tables 7 to 9.

From the J30 results obtained at each comparison point (1000, 5000 and 50000 fitness evaluations), COA was ranked 1^{st}. Also, it was interesting that the average deviation it obtained in 1000 schedules was better than that of many other

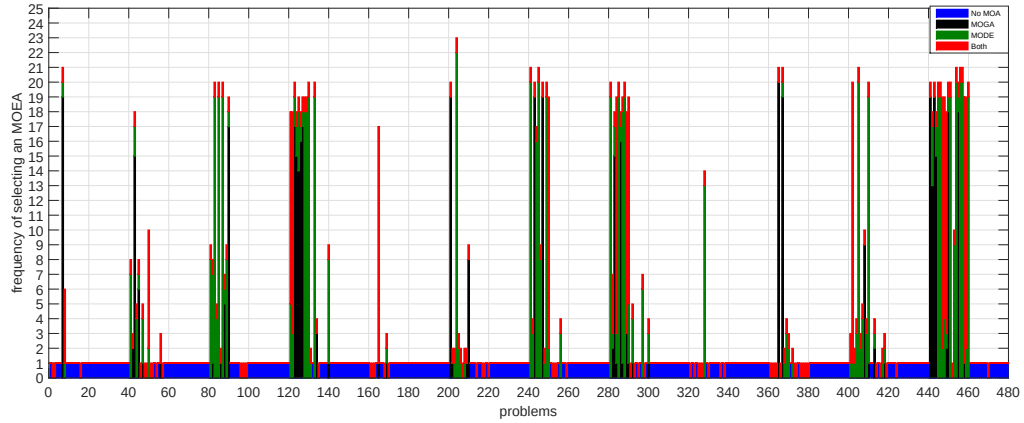


(a)

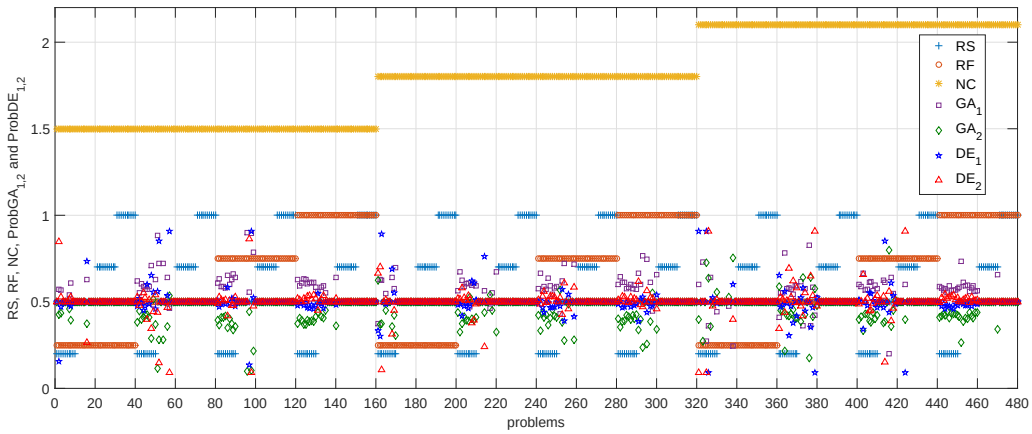


(b)

Figure 6: (a) frequency of use of each optimization algorithm and (b) and average probability of each operator for J30 activities

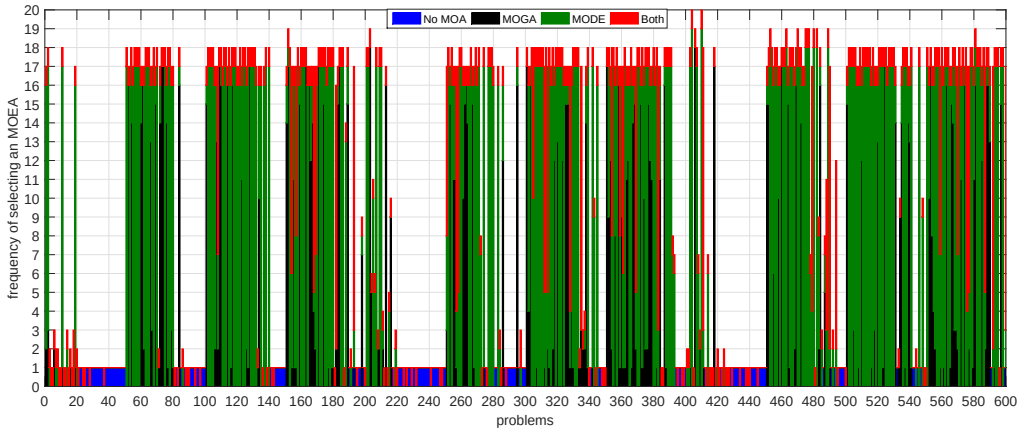


(a)

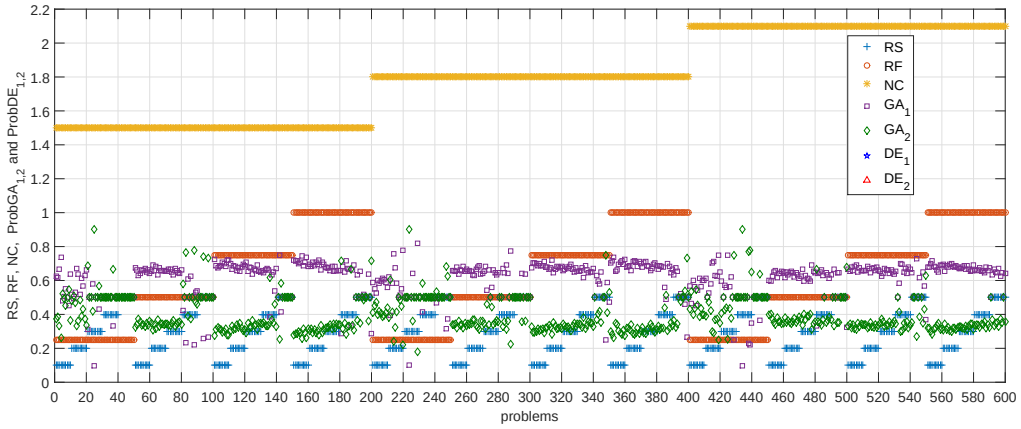


(b)

Figure 7: (a) frequency of use of each optimization algorithm and (b) and average probability of each operator for J60 activities



(a)



(b)

Figure 8: (a) frequency of use of each optimization algorithm and (b) and average probability of each operator for J120 activities

Table 7: Average deviation values obtained by COA and other state-of-the-art algorithms for J30

Algorithms	1000	5000	50000
COAs(this study)	**0.04**	**0.00**	**0.00**
GA-MBX [42]	0.14	0.04	**0**
GA [30]	0.06	0.02	0.01
ACO+SS [11]	0.14	0.06	0.01
PSO-HH [28]	0.26	0.04	0.01
DBGA [16]	0.12	0.04	0.02
HGA [40]	0.27	0.06	0.02
PSO [10]	0.29	0.14	0.04
BA [46]	0.42	0.19	0.04
SFLA [20]	0.36	0.21	0.18
HEDA [41]	0.38	0.14	-
GANN [1]	0.13	0.1	-

Table 8: Average deviation values obtained by COA and other state-of-the-art algorithms for J60

Algorithms	1000	5000	50000
COAs (this study)	**11.13**	**10.77**	**10.58**
GA-MBX [42]	11.33	10.94	10.65
SFLA [20]	11.44	10.87	10.66
GA [30]	11.72	11.04	10.67
ACO+SS [11]	11.75	10.98	10.67
DBGA[16]	11.31	10.95	10.68
PSO-HH [28]	11.74	11.13	10.68
HGA [40]	11.56	11.1	10.73
PSO [10]	12.03	11.43	11
BA [46]	12.55	12.04	11.16
HEDA [41]	11.97	11.43	-
GANN [1]	11.51	11.29	-

algorithms even when they used a higher number of fitness evaluations. This algorithm performed consistently for the J60 problems with a large difference in deviations compared with those of the other algorithms, especially for smaller numbers of fitness evaluations.

However, considering the J120 results, COA was outperformed by 3 other algorithms at the maximum number of schedules, and ranked 3^{rd} at 1000. Regarding $\text{COAs}_{Adaptive}\text{J}_{120,0.025-0.001}$, which was designed to accelerate the convergence rate (Section 4.2), the results showed that COA was ranked 2^{nd} at 1000 and 5000 fitness evaluations, and 4^{th} at 50000.

4.5. Effect of number of multi-operator meta-heuristics

In this section, the effect of the number of multi-operator meta-heuristics is analyzed. 5 different variants of COA were run with the first three using a single multi-operator meta-heuristic, i.e., MOGA ($COA_{1,MOGA}$) , MODE ($COA_{2,MODE}$), and MOPSO ($COA_{1,MOPSO}$), the 4^{th} (COA_4) the version discussed above, which used MOGA and MODE, and the last (COA_5) used three multi-operator meta-heuristics (MOGA, MODE and MOPSO).

From the results, it was found that $\text{COA}_{1,MOGA}$ and $\text{COA}_{2,MODE}$ were better than $\text{COA}_{3,MOPSO}$, and that COA_4 was better than the individual algorithms used in it. For J30, although COA_5 obtained better results than COA, its results were inferior to those of $\text{COA}_{1,MOGA}$ and $\text{COA}_{2,MODE}$. The reason for this might have been that, as one of the algorithms used in COA_5, MOPSO, performed very poorly and consumed many fitness evaluations, the other multi-operator meta-heuristics did not have sufficient fitness evaluations to evolve compared with those used in the other variants. COA_5 did better for J60 than J30, and obtained better results than $\text{COA}_{2,MODE}$ and $\text{COA}_{3,MOPSO}$ at 5000 and 50000 fitness evaluations, respectively, and almost the same as $\text{COA}_{1,MOGA}$ for 50000. Generally, the framework worked efficiently if the independent multi-operator meta-heuristics used in it were complementary, which is consistent with the conclusion in [33]. From the results obtained, the variant with two multi-operator meta-heuristics (COA_4) was considered the best.

Table 9: Average deviation values obtained by COA and other state-of-the-art algorithms for J120

Algorithms	1000	5000	50000
ACO+SS [11]	35.19	32.48	**30.56**
DBGA[16]	**33.55**	**32.18**	30.69
SFLA [20]	34.83	33.2	31.11
COAs (default)	34.04	32.90	31.22
$\mathbf{COAs}_{\mathbf{J}120,Adaptive_{0.025-0.001}}$	33.68	32.35	31.23
PSO-HH [28]	35.2	32.59	31.23
HGA [40]	34.07	32.54	31.24
GA-MBX [42]	34.02	32.89	31.3
GA [30]	35.87	33.03	31.44
PSO [10]	35.71	33.88	32.89
BA [46]	37.72	36.76	34.55
HEDA [41]	35.44	33.61	-
GANN [1]	34.65	34.15	-

Table 10: Average deviation percentages obtained by different variants of COA for J30 and J60

	Variants	J30		J60	
		Schedules		Schedules	
		5000	50000	5000	50000
$\overline{Dev}$	$COA_{1,MOGA}$	0.02	**0.00**	10.78	10.61
	$COA_{2,MODE}$	0.01	**0.00**	11.06	10.71
	$COA_{3,MOPSO}$	0.13	0.01	11.24	10.79
	COA_4	**0.00**	**0.00**	**10.77**	**10.58**
	COA_5	0.09	0.01	10.82	10.61

5. Conclusions and Future Work

Although many methods for solving RCPSPs have been introduced, they still have some shortcomings, such as a slow rate of convergence, and no single optimization one has proven to be the best for all problems. Even in a single algorithm, it is tedious to select its set of operators that can perform consistently well over a wide range of RCPSPs. As a result, in this paper, a new framework which could be considered a step toward better optimization algorithms was proposed. The algorithm (COAs) utilized the strengths of two EAs (GA and DE), each ran with two different crossover or mutation operators, respectively. During the evolutionary process, a mechanism was adopted to place emphasis on the best-performing algorithm as well as its operators. Also, the benefit of an efficient heuristic was utilized to further enhance its performance.

This algorithm was used to solve well-known benchmark problems, with 30, 60 and 120 activities, with he results showing that it was capable of converging quickly to high-quality solutions, especially for the J30 and J60 problems. Three parameters (CS, PS and MR_{GA}) were analyzed. The first, the only new one introduced in the proposed design, did not demonstrate a significant effect on the algorithm's performance, while the other two did which led to the following conclusions being drawn.

- It was recommended that a small value of PS, i.e., 10, be used to achieve a high convergence rate. However, if the decision maker was concerned with only the quality of solutions for 50000 schedules, then $PS = 50$ would be the best.

- MR_{GA} had a significant effect on the algorithm's performance, with every small value leading to low-quality solutions, while high ones produced slow convergence rates. Generally, adaptively reducing it from 0.05 to 0.001 achieved the best performances for J30 and J60.

- For J120, to further accelerate the convergence rate, adaptively reducing MR_{GA} from 0.025 to 0.001 was best. The reason for using a different MR_{GA} than that used for the other dimensions was that the characteristics of J120 were different and there was the curse of dimensionality.

We also tried to answer a question: *which algorithm and operator did work well for the RCPSPs considered*? The answer was consistent with the motivation of the paper, in which no single algorithm and/or operator were the best for all types of RCPSPs. Also, it was confirmed that one algorithm and/or operator might be good in some generations of the evolutionary process and poor in others. However, some clues were found: (**1**) for J30, MOGA performed well when

RC was small and $NC = 1.5$. In the case of $NC \leq 1.8$ and RC high, both MOGA operators were performing almost the same; (**2**) for J60, MODE was more favorable than MOGA, with DE_1 slightly more preferable when $RC = 0.5$ and NC was small, the opposite when NC was high, and both quite similar when $RC = 1.0$. For MOGA, GA_2 had a higher probability when NC was high and RC small; and (**3**) for J120, MODE was better for solving the majority of test problems, with DE_1 and DE_2 having close possibilities of running, and for MOGA, GA_1 dominated GA_2 for the majority of test problems.

As COAs preferred MODE more to MOGA for many J60 and J120 problems, researchers may think of conducting more DE research on scheduling problems, especially when we found limited research studies in the literature that used DE to solve RCPSPs.

The results were compared with those from the state-of-the-art algorithms in terms of the quality of solutions and showed the best performances for the J30 and J60 problems, and were very competitive for J120. Also, because the GA operators were not complementary when solving J120, i.e., GA_1 dominated GA_2 for most of the test problems, the algorithm did not demonstrate as consistent performances those that for J30 and J60.

Also, it was found that two multi-operator meta-heuristics performed better than variants with one or three sub-algorithms. In general, the proposed algorithm could work efficiently if the independent sub-algorithms used in it complemented each others.

In future work, further investigations are needed to accelerate the algorithm's convergence and quality of solutions for the J120 problems. Combining other EAs and heuristics in one framework will be an interesting aspect and solving multi-mode RCPSPs will be considered.

Acknowledgment

This research is supported by the Australian Research Council Discovery Project DP150102583 awarded to R. Sarker and CONACyT project no. 221551 awarded to C. Coello Coello.

References

[1] Agarwal, A., Colak, S., Erenguc, S., 2011. A neurogenetic approach for the resource-constrained project scheduling problem. Computers & Operations Research 38 (1), 44–50.

[2] Alcaraz, J., Maroto, C., 2001. A robust genetic algorithm for resource allocation in project scheduling. Annals of Operations Research 102 (1-4), 83–109.

[3] Ali, I., 2016. Evolutionary algorithms for resource constrained project scheduling. Master's thesis, Engineering and Information Technology, UNSW Canberra.

[4] Ali, I. M., Elsayed, S. M., Ray, T., Sarker, R. A., 2015. Memetic algorithm for solving resource constrained project scheduling problems. In: IEEE Congress on Evolutionary Computation. IEEE, pp. 2761–2767.

[5] Ali, I. M., Elsayed, S. M., Ray, T., Sarker, R. A., 2016. A differential evolution algorithm for solving resource constrained project scheduling problems. In: Australasian Conference on Artificial Life and Computational Intelligence. Springer, pp. 209–220.

[6] Anagnostopoulos, K., Koulinas, G., 2011. Resource-constrained critical path scheduling by a grasp-based hyper-heuristic. Journal of Computing in Civil Engineering 26 (2), 204–213.

[7] Anagnostopoulos, K. P., Koulinas, G. K., 2010. A genetic hyperheuristic algorithm for the resource constrained project scheduling problem. In: IEEE Congress on Evolutionary Computation. IEEE, pp. 1–6.

[8] Artigues, C., Demassey, S., Neron, E., 2013. Resource-constrained project scheduling: models, algorithms, extensions and applications. John Wiley & Sons.

[9] Bartels, J.-H., Zimmermann, J., 2015. Scheduling tests in automotive r&d projects using a genetic algorithm. In: Handbook on Project Management and Scheduling Vol. 2. Springer, pp. 1157–1185.

[10] Chen, R.-M., 2011. Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem. Expert Systems with Applications 38 (6), 7102–7111.

[11] Chen, W., Shi, Y.-j., Teng, H.-f., Lan, X.-p., Hu, L.-c., 2010. An efficient hybrid algorithm for resource-constrained project scheduling. Information Sciences 180 (6), 1031–1039.

[12] Cheng, M.-Y., Tran, D.-H., Wu, Y.-W., 2014. Using a fuzzy clustering chaotic-based differential evolution with serial method to solve resource-constrained project scheduling problems. Automation in Construction 37, 88–97.

[13] Christofides, N., Alvarez-Valdés, R., Tamarit, J. M., 1987. Project scheduling with resource constraints: A branch and bound approach. European Journal of Operational Research 29 (3), 262–273.

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

[15] Deb, K., 2000. An efficient constraint handling method for genetic algorithms. Computer methods in applied mechanics and engineering 186 (2), 311–338.

[16] Debels, D., Vanhoucke, M., 2007. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. Operations Research 55 (3), 457–469.

[17] Demeulemeester, E., Herroelen, W., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management science 38 (12), 1803–1818.

[18] Eberhart, R. C., Kennedy, J., et al., 1995. A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science. Vol. 1. New York, NY, pp. 39–43.

[19] Elsayed, S. M., Sarker, R. A., Essam, D. L., 2011. Multi-operator based evolutionary algorithms for solving constrained optimization problems. Computers & operations research 38 (12), 1877–1896.

[20] Fang, C., Wang, L., 2012. An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem. Computers & Operations Research 39 (5), 890–901.

[21] Gonçalves, J. F., Resende, M. G., Mendes, J. J., 2011. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Journal of Heuristics 17 (5), 467–486.

[22] Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127 (2), 394–407.

[23] Jia, Q., Seo, Y., 2013. An improved particle swarm optimization for the resource-constrained project scheduling problem. The International Journal of Advanced Manufacturing Technology 67 (9-12), 2627–2638.

[24] Kolisch, R., 1996. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management 14 (3), 179–192.

[25] Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research 90 (2), 320 – 333.

[26] Kolisch, R., Schwindt, C., Sprecher, A., 1999. Benchmark instances for project scheduling problems. In: Project Scheduling. Springer, pp. 197–212.

[27] Kolisch, R., Sprecher, A., 1997. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. European journal of operational research 96 (1), 205–216.

[28] Koulinas, G., Kotsikas, L., Anagnostopoulos, K., 2014. A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. Information Sciences 277, 680–693.

[29] Li, K., Willis, R., 1992. An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research 56 (3), 370–379.

[30] Mendes, J. J. d. M., Gonçalves, J. F., Resende, M. G., 2009. A random key based genetic algorithm for the resource constrained project scheduling problem. Computers & Operations Research 36 (1), 92–109.

[31] Merkle, D., Middendorf, M., Schmeck, H., 2002. Ant colony optimization for resource-constrained project scheduling. IEEE transactions on evolutionary computation 6 (4), 333–346.

[32] Messelis, T., De Causmaecker, P., 2014. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. European Journal of Operational Research 233 (3), 511–528.

[33] Peng, F., Tang, K., Chen, G., Yao, X., 2010. Population-based algorithm portfolios for numerical optimization. IEEE Transactions on Evolutionary Computation 14 (5), 782–800.

[34] Sarker, R., Kamruzzaman, J., Newton, C., 2003. Evolutionary optimization (evopt): a brief review and analysis. International Journal of Computational Intelligence and Applications 3 (04), 311–330.

[35] Socha, K., Dorigo, M., 2008. Ant colony optimization for continuous domains. European journal of operational research 185 (3), 1155–1173.

[36] 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.

[37] Tanabe, R., Fukunaga, A., 2013. Success-history based parameter adaptation for differential evolution. In: IEEE Congress on Evolutionary Computation. IEEE, pp. 71–78.

[38] 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.

[39] Valls, V., Ballestín, F., Quintanilla, M., 2002. A hybrid genetic algorithm for the rcpsp with the peak crossover operator. In: Proc. of 8th International Workshop on Project Management and Scheduling, PMS 2002. Citeseer, pp. 368–370.

[40] Valls, V., Ballestin, F., Quintanilla, S., 2008. A hybrid genetic algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 185 (2), 495–508.

[41] Wang, L., Fang, C., 2012. A hybrid estimation of distribution algorithm for solving the resource-constrained project scheduling problem. Expert Systems with Applications 39 (3), 2451–2460.

[42] Zamani, R., 2013. A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. European Journal of Operational Research 229 (2), 552–559.

[43] Zhang, H., Li, H., Tam, C., 2006. Particle swarm optimization for resource-constrained project scheduling. International Journal of Project Management 24 (1), 83–92.

[44] Zhang, H., Li, X., Li, H., Huang, F., 2005. Particle swarm optimization-based schemes for resource-constrained project scheduling. Automation in Construction 14 (3), 393–404.

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

[46] Ziarati, K., Akbari, R., Zeighami, V., 2011. On the performance of bee algorithms for resource-constrained project scheduling problem. Applied Soft Computing 11 (4), 3720–3733.