# A Forward-Backward Relax-and-Solve Algorithm for the Resource-Constrained Project Scheduling Problem

Alireza Etminaniesfahani[1*], Hanyu Gu[1], Leila Moslemi Naeni[2] and Amir Salehipour[3]

[1*]School of Mathematical and Physical Sciences, University of Technology Sydney, Broadway, Sydney, 2007, NSW, Australia.
[2]School of the Built Environment, University of Technology Sydney, Broadway, Sydney, 2007, NSW, Australia.
[3]The University of Sydney Business School, Darlington, 2006, NSW, Australia.

*Corresponding author(s). E-mail(s): Alireza.Etminaniesfahani@student.uts.edu.au;
Contributing authors: Hanyu.Gu@uts.edu.au; Leila.MoslemiNaeni@uts.edu.au; amir.salehipour@sydney.edu.au;

**Abstract**

Scheduling projects under limited resource availability, which is called the resource-constrained project scheduling problem (RCPSP), has a wide range of real-world applications, e.g., in mining, manufacturing and supply chain. The RCPSP is NP-hard, and over the last five decades researchers attempted to propose various solution techniques for this challenging problem. The relax-and-solve (R&S) algorithm is a recently proposed method for solving various scheduling problems such as job-shop and single and parallel machine scheduling problems. This research contributes to the existing research on the R&S by presenting an easy-to-implement and effective R&S method for solving RCPSP. Our R&S employs CPLEX CP optimizer as an optimization solver to generate and optimize schedules within a heuristic framework. We further improve the algorithm's performance by employing forward-backward passes. The results of testing the algorithms on 1560 standard instances from the well-known PSPLIB show our heuristic delivers competitive results and outperforms state-of-the-art methods for solving the RCPSP.

**Keywords:** Relax-and-Solve, Resource-Constrained Project Scheduling Problem, Matheuristic, Forward-backward improvement

# 1 Introduction

Scheduling a large project under several precedence and resource constraints has always been challenging for project managers. A common source of delays in projects is known to be lack of an efficient project scheduling technique [1]. For more than 70 years, the critical path method (CPM) has been one of the most extensively used planning and scheduling approaches. CPM assumes unlimited resources are always available. Therefore, by considering precedence constraints the CPM may provide a lower bound for the project duration.

The resource-constrained project scheduling problem (RCPSP) was introduced by Pritsker et al. in 1969 [2]. In addition to precedence constraints, the authors considered the restriction of resource availability over time. Since then the RCPSP and its extensions have been used in a wide range of practical applications in diverse industries such as supply chain [3], mining [4] and job-shop scheduling [5]. The RCPSP's vast range of applications, as well as its computational challenges (it is an NP-Hard problem [6]) has attracted the attention of many scholars [7]. The RCPSP's solution methods are generally divided into exact algorithms and heuristics-based approaches [8]. While the optimality of the solutions in exact methods is guaranteed, it is, however, at the expense of increasing solution time, which makes exact methods impracticable in solving large-scale problems [9]. Approaches based on heuristics can overcome the shortcoming of exact methods in computation time, although obtaining an optimal solution cannot be guaranteed [10].

Generally, heuristic methods solve RCPSP in two steps [11]. In the first step, a task list is provided based on heuristic rules, and a rank is assigned to each task based on their priorities. In the second step, considering both precedence and resource constraints, the start and finish times of tasks are determined. In this step, two different methods for transferring the list from step 1 can be employed, namely forward loading and backward loading [11]. In the forward loading method, which generates schedules in a forward pass (e.g., starting from time zero and moving ahead), tasks are started as early as possible, i.e., as soon as all of their precedence constraints are satisfied and adequate resources are available. In contrast, the backward loading method generates schedules in a backward pass in which each task is selected as late as possible subject to not exceeding the target finish time of the project. Schedules produced by employing backward loading methods can be more efficient than those employing forward loading methods [11].

The forward-backward improvement (FBI) [11] is an effective method to generate quality feasible schedules. In the forward pass a feasible schedule is generated in which each task starts as early as possible. In the backward pass tasks are started at the latest time such that all constraints are satisfied and

the target finish time of the project is met. Iterative scheduling in the forward and backward passes can improve the results[11]. This procedure terminates when there is no improvement in two successive solutions obtained by forward and backward passes. According to [10], FBI can be hybridized with heuristic algorithms. In fact, many of the best methods in solving RCPSP employ this method [12].

The genetic algorithm (GA) [13] is one of the most used methods to solve the RCPSP [14]. Nonetheless, several studies used FBI with GA [14–24]. [25] improved the evolutionary algorithm by FBI in EA(FBI). [26] developed combined FBI and the ant colony optimization (ACO) [27] to solve RCPSP, and [28] implemented three algorithms of bee algorithm (BA) [29], artificial bee colony (ABC) [30] and bee swarm optimization (BSO) [31]. They improved these methods by a new constraint handling method and a new local search. They also implemented the variants of the algorithms in which the algorithms were combined with the FBI: BSO-FBI, ABC-FBI, and BA-FBI. The results show that incorporating FBI leads to quality schedules. [32] used FBI with a scatter search metaheuristic. In each iteration, the algorithm reverses the project network. A hybrid FBI with Lagrangian relaxation (LR-FBI) proposed by [33].

The constraint programming (CP) has successfully been used to solve scheduling problems [34–36] including the RCPSP [37–40].

The general R&S, including the one proposed in the present research has significant differences than relax-and-fix and fix-and-optimize methods. The relax-and-fix method [41] defines a rolling time window to separate the binary variables to be fixed and those to be optimized (their values are decided upon). This algorithm relaxes the integrality constraint of those variables that are not in the rolling time window. The fix-and-optimize method [42, 43] deals with the problem by operating on two sets of variables, namely fixed and optimized. The main advantage of this algorithm is that it always provides feasible solutions through maintaining integrality constraints.

The R&S algorithm has recently been proposed for various scheduling problems such as job-shop and single and parallel machine scheduling problems [44–48].

We are motivated to conduct the present research due to the fact that RCPSP is an NP-hard problem [6], which means only small-sized instances of RCPSP can be solved by using exact methods in a reasonable computational time [49]. Although there are various open-source [50] and commercial solvers such as CPLEX CP optimizer [51] that can be used for solving optimization problems, including RCPSP, however, the efficiency of those solvers typically decreases when the size of instances increases (e.g., solving an instance of RCPSP with more than 100 tasks with those solvers is very challenging). In general, for solving challenging instances with more than 30 tasks various heuristic-based solution techniques have been proposed over the last five decades. That along with inefficiency of the solvers motivated us to propose a

method that employs the CPLEX CP optimizer within the R&S matheuristic that is able to effectively and efficiently solve RCPSP.

The R&S method proposed in the present paper to solve RCPSP first generates a feasible schedule (a solution). Then, it improves the feasible solution in an iterative process, which includes two main phases, namely relaxing phase and solving phase. The algorithm relaxes the execution order of a few tasks at a time in a solution to generate a relaxed problem. Then, the algorithm solves the whole problem by a solver, which leads to developing a schedule of executing the tasks.

Relaxing the problem in this way helps to reduce the solution time because smaller problems need to be solved at every iteration. The main challenging part of employing such relaxation method in solving RCPSP includes defining the set of task in a schedule to be relaxed. To handle that issue, we use a rolling time window to distinguish between tasks that are inside the time window (permitted to be reordered, i.e., relaxed), and the tasks that are located outside the time window. To relax the problem, when completing time of one task and starting time of another task are the same, the algorithm adds start-at-end constraints between them. In this way, the algorithm prevents the change in the relative order of tasks outside the time window.

The contributions of our work include (*i*) developing a novel matheuristic algorithm for RCPSP with forward and backward passes, (*ii*) employing CPLEX CP optimizer to create schedules in both forward and backward passes, whereas the CPLEX CP optimizer is usually employed to generate schedules in the forward pass, (*iii*) proposing a novel and efficient technique to develop relaxed problems at each step using the current solution, and (*iv*) delivering competitive solutions for instances from the PSPLIB (project scheduling problem library) with 30, 60 and 120 activities.

In what follows, we formally define RCPSP (Section 2), and provide our forward-backward R&S matheuristic in Section 3. In Section 4, we report the results of our computational experiments, and in Section 5 we provide conclusions and future research directions.

## 2 Problem definition and formulation

The RCPSP includes a set of $n$ task. Task $i$ has a certain non-negative duration presented by $d_i$. Precedence constraints also exist between tasks that can be modeled as an activity-on-node network $G = (V, A)$. Every single task relates to a node in the node set $V = \{1, 2, \ldots, n+1\}$. Arc $(i, j)$ in the arc set $A$ represents the precedence constraint that task $i$ has to complete before task $j$ can start. Dummy nodes 0 and $n+1$ are added to the set of nodes to represent the project's start and finish times, respectively. The duration of dummy nodes (tasks), i.e., $d_0$ and $d_{n+1}$ are 0. Because time cannot reverse the graph $G$ is always acyclic. A fixed set of renewable resources represented by $RR$ is available. Every single resource $k \in RR$ has a non-negative and fixed

capacity $R_k$ at each time in planning horizon $T$. Each task $i$ needs a non-negative amount of $r_{ik}$ of each resource $k \in RR$. Tasks cannot be interrupted once started (non-preemptive).

The mathematical formulation of the RCPSP can be expressed as follows, where $S_i$ presents the starting time of the task $i$. The first dummy task starts at $S_0 = 0$, and the finishing time of the project (makespan) is the finishing time of the last dummy job. Because the duration of dummy jobs is equal to zero, the project makespan is then equal to $S_{n+1}$.

$$\min S_{n+1} \tag{1}$$

subject to

$$S_j \geq S_i + d_i, \quad \forall (i,j) \in A, \tag{2}$$

$$\sum_{i \in \tau(t)} r_{ik} \leq R_k, \forall k \in RR, \forall t \in \{0, \ldots, T-1\}, \tag{3}$$

$$\tau(t) = \{i \in V \mid S_i \leq t < S_i + d_i\}, \tag{4}$$

$$S_i \in \{0, 1, \ldots, T - d_i\}, \quad \forall i \in V, \tag{5}$$

where $T$ is the given upper bound on the project duration.

# 3 The proposed forward-backward relax-and-solve method

This section proposes an efficient forward-backward R&S matheuristic algorithm for the RCPSP. The main idea of this method is to reduce the solution time by reducing the complexity of the problem at every iteration. We aim to relax the problem in a way that all tasks are involved in the solution while taking the advantage of the forward-backward method to improve the searchability of the algorithm. In the proposed forward-backward R&S algorithm an initial feasible solution is generated and gradually improved like most heuristics. For a feasible schedule a rolling time window is defined for the "relax" phase, in which all tasks outside the window are "glued" with each other with respect to the order in the current solution.

Only tasks inside the time window can be reordered. In the "solve" phase, a feasible schedule is obtained by solving the relaxed problem. As we do not remove the precedence and resource constraints the obtained solution is always feasible for the problem.

CPLEX CP optimizer is used to generate a solution in the forward pass considering problem's resource and precedence constraints. In addition to the forward pass, we generate a solution in a reversed direction i.e., in a backward pass. To this aim, the precedence constraints should be reversed, which means

each arc $(i, j)$ should be changed to arc $(j, i)$. By doing so tasks are scheduled in a reversed order starting from the last task.

The algorithm starts with the initial solution generated in the backward pass. In each iteration, the algorithm relaxes certain tasks in the problem, which results in a relaxed problem, and then solves the relaxed problem in the forward pass. At the end of each iteration, the algorithm removes all added start-at-end constraints and solves the problem in the backward pass for a limited time to avoid being trapped in a local optimum.

The general forward-backward R&S algorithm is summarized in Algorithm 1.

**Algorithm 1** The forward-backward R&S algorithm.

**Input:** A RCPSP instance.
**Output:** A feasible schedule.
Generate an initial feasible solution using CPLEX CP optimizer on the backward pass formulation of the instance (see Section 3.1).
**Set** time window starting time $st = 0$
**while** *the stopping condition is not met* **do**
  **if** $st > C_{\max} - L$ **then**
    $st = 0$ ▷ $C_{\max}$ is the makespan and $L$ is the time window length
  **end**
  Determine the tasks of group 1 and the tasks of group 2 (see Sections 3.2.2, and 3.2.3);
  Generate a relaxed problem (see Section 3.2);
  Solve the relaxed problem in the forward pass by using the CPLEX CP optimizer (Section 3.3.1);
  $st = st + L - overlap$; ▷ *overlap* is a parameter
  Solve the original problem in the backward pass with CPLEX CP optimizer (Section 3.3.2);
**end**
**return** *the best obtained schedule (the solution);*

In what follows we discuss: generating an initial solution for the problem; generating and solving a relaxed problem in each iteration; and setting the stopping criterion for the algorithm.

## 3.1 Initial solution

[52] discussed the effectiveness of the CP in generating initial solutions for a crew scheduling problem. CPLEX CP optimizer, which is a commercial solver for CP, can generate quality solutions for challenging combinatorial optimization problems [53]. We use CPLEX CP optimizer in the backward pass to generate an initial solution.

To solve RCPSP, most approaches prefer to operate on the representation of the solutions and then employ decoding methods to transform those representations into a schedule. Common representation methods for RCPSP were

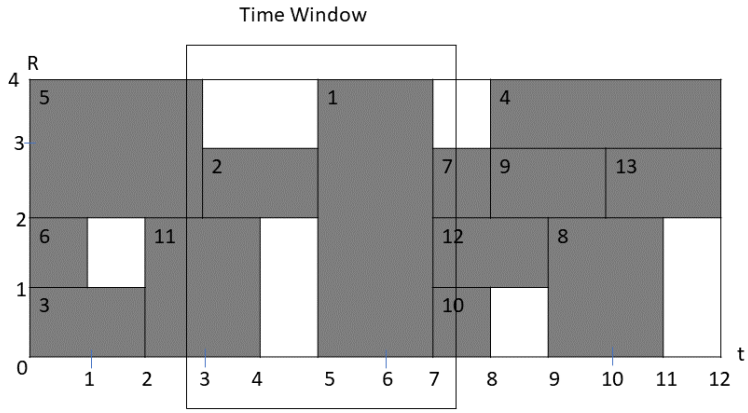


**Fig. 1**: An example demonstrating how time window is defined for RCPSP.

proposed in [54]. In the present paper, we operate on schedules rather than representations. Therefore, the solution for an instance is proposed through starting time for each task.

## 3.2 Generating the relaxed problem

Figure 1 illustrates an example of a feasible solution that will be relaxed in solving an RCPSP using the forward-backward R&S method. A rolling time window is employed to specify which part of a problem is relaxed.

In the figure, the time window divides the tasks into group 1 and group 2. In what follows we explain how the rolling time window works and how the tasks of each group are treated to generate a relaxed problem.

### 3.2.1 The rolling time window

A time window is utilized to relax a schedule. We assume there is a rolling frame on the time horizon of an existing schedule. We provide an example in Figure 1 to show how a rolling time window specifies which part of the problem should be relaxed. Generally, increasing the length of the time window increases the number of tasks that are free to be reordered. The time window's starting time is set as $t = 0$, and after each iteration it moves forward at a specific rate. As soon as the time window reaches the end of the time horizon it is set to $t = 0$. While moving the time window forward, a critical parameter preventing the algorithm from being trapped in local optima is the *overlap*, which is the amount of overlap between time windows in successive iterations. If there is no overlap, then the tasks are restricted to be inside a time window; hence, those tasks cannot move to other time windows. We suggest to define the length of the time window and *overlap* between time windows proportion to the makespan of the current feasible schedule of the problem that is subject

to relaxing. Later in section 4, we propose to set the value of parameter *overlap* between 0 and 1.

We propose to calculate the length of the time window $L$ as a function of the number of generated relaxed problems i.e., $N$, and also *overlap*, and makespan i.e.,$C_{\max}$ , in particular $L = (\frac{C_{\max}}{N}) \times (1 + overlap)$.

### 3.2.2 Tasks of group 1 ($G1$)

The tasks that finish before the start of the time window, or start after the finish of the time window, i.e., those tasks that are entirely beyond the time window belong to group 1. For example, in Figure 1, the set of tasks of group 1 includes $G1 = \{3, 6, 4, 9, 8, 13\}$. The algorithm does not allow the tasks of group 1 to change their relative execution order. That is the main idea behind relaxation as we define and use in the present paper.

That can be achieved by adding "start-at-end" constraints between the tasks, which states that the start time of one task is equal to the finish time of another task. Examples include $(3, 11)$ or $(12, 8)$ in Figure 1. In this example, tasks $7, 4, 9, 13$ are also fixed together.

A point that should be noted here is that one task starts as soon as its resource constraints and precedence constraints are satisfied. So each task starts at $t = 0$ or starts at the finish time of another task. In such a case, as for task 6 in the example, the algorithm does not need to fix the task with other tasks

### 3.2.3 Tasks of group 2 ($G2$)

The task of group 2 or $G2$ are those tasks that are completely or partially inside the time window. For example, $G2 = \{5, 11, 2, 1, 7, 12, 10\}$ in Figure 1 shows tasks of group 2. The tasks of this group can be reordered subject to satisfying all problem's constraints, if that leads to an improved schedule

## 3.3 Solving Phase

This phase includes two steps: 1) solving each relaxed problem, and 2) solving the original problem. In both steps, the forward-backward R&S algorithm uses the CPLEX CP optimizer.

### 3.3.1 Solving the relaxed problem

The relaxed problem includes a fewer tasks to be reordered compared to the original problem, which results in the relaxed problem to be relatively easier to solve. The obtained solution to the relaxed problem is always feasible because the tasks of group 1 cannot be reordered, however, the starting time of each set of fixed tasks is flexible, so the makespan of the relaxed problem is always the makespan of the original problem. We solve each relaxed problem in the forward pass.

### 3.3.2 Solving the original problem

The algorithm solves the general problem immediately after solving each relaxed problem. Solving the original problem even for a few seconds in backward pass improves the global exploration of the algorithm. Therefore, it is beneficial to search the space in both directions.

## 3.4 Stopping criterion

The maximum number of iterations should be determined before solving a problem. In each iteration, a time limit is set for the CPLEX CP optimizer to solve each forward or backward pass. Once the CPLEX CP optimizer obtains the optimal solution, or if the computation time reaches the limit, the algorithm goes to the next iteration. The algorithm terminates when it reaches the maximum number of iterations.

# 4 Computational experiments

In this section, we provide computational results of the proposed forward-backward R&S on instances from the well-known PSPLIB [55]. We consider instance sets J30, J60 and J120, which include 30, 60, and 120 tasks, respectively. Instance sets J30 and J60 include 480 instances and the set J120 includes 600 instances. We coded the algorithm by using the Python programming language version 3.6.5 and we use CPLEX CP optimizer version 12.10.0.0 [51].

Unless otherwise stated we use default value for CP parameters. We conduct all the experiments on a machine with CPU Intel Xeon Gold 6238R 2.2GHz.

Next, we discuss the parameters setting followed by the detailed computational results.

## 4.1 Parameters setting

The initial solution generated in a backward pass is obtained by setting the time limit to 30 seconds. The stopping time for solving each relaxed problem is set to 60 seconds. The allocated time for solving the original problem in the backward pass (at the end of each iteration) is set to 30 seconds. Those lead to average computational times of less than 150, 300 and 600 seconds, for instance sets J30, J60, and J120, respectively. We set $overlap = 0.33$ and the maximum number of relaxed problems $N = 0.1 \times n$.

Each relaxed problem should be solved in one iteration, meaning that the maximum number of iterations is equal to 3, 6, and 12 for J30, J60, and J120, respectively.

## 4.2 Results

In this section, we compare our results and those obtained by CPLEX CP optimizer and three exact methods including failure directed search (FDS)[56],

**Table 1**: comparison of LCG, FDS, SMT, CPLEX, and forward-backward R&S .

| | J30 | | J60 | | J120 | |
|---|---|---|---|---|---|---|
| Method | $\Delta_{LB}$ | $t$ | $\Delta_{LB}$ | $t$ | $\Delta_{LB}$ | $t$ |
| LCG | 0 | - | 2.17 | - | 9.76 | - |
| FDS | 0 | 0.93s | 1.91 | 67.44s | 7.02 | 322.52s |
| SMT | 0 | 0.22s | 1.88 | 61.90s | 9.55 | 320.50s |
| CPLEX CP optimizer | 0 | 4.71s | 1.11 | 52.83s | 4.69 | 350.40s |
| R&S | 0 | 5.12s | 1.06 | 46.75s | 4.63 | 235.65s |
| Forward-backward R&S | 0 | 7.17s | 1.06 | 66.21s | 4.61 | 200.11s |

lazy clause generation (LCG) [57], satisfiability modulo theories (SMT) by [58], that shown competitive results in solving RCPSP.

In order to compare the results delivered by the proposed method and the exact methods we present a summary of the results in Table 1. In this table, $t$ is the computation time in seconds and the average deviation from the best known lower bound for each instance is calculated as:

$$\Delta_{LB} = \frac{\sum_{p=1}^{p=P} \frac{C_{max,forward-backwardR\&S,p} - C_{max,LB,p}}{C_{max,LB,p}} \times 100\%}{P}, \tag{6}$$

where $P$ is the number of instances in each set, $C_{max,forward-backwardR\&S,p}$ is the makespan obtained by Algorithm 1, and $C_{max,LB,p}$ is the best known lower bound on the makespan for each instance.

For each method, the computation time and the average deviation from the best known lower bound are provided in Table 1. Results show that employing the forward-backward method in R&S improves the performance of R&S, leading to superior results, in terms of $\Delta_{LB}$ criterion, to other methods for solving J60 and J120 instances. All of the tested methods obtain optimal solution for J30. Comparing the result of the CPLEX CP optimizer, as a stand-alone method, and both R&S and forward-backward R&S for J60 and J120 illustrates the efficiency of the forward-backward R&S in solving the tested instances. For almost all instances of J30, the CPLEX CP optimizer finds the optimal solution in a short time.

In Table 2, we compare the results of forward-backward R&S and the state-of-the-art metaheuristics as in [7], including memetic algorithm (MA), consolidated optimization algorithm (COA) [65], PSO-based hyper-heuristic algorithm (PSO-HH) [66] and genetic algorithm using forward-backward improvement (GAFBI) [64], and those algorithms that combine the forward-backward method with GA [14, 15, 17–24], PSO [59, 62, 63], ACO [26], ABC [28], the Lagrangian relaxation [33] and scatter search [32].

In the literature, it is common to set the number of generated schedules as the stopping criterion. However, it does not apply to our algorithm because it

**Table 2**: Comparison of the state-of-the-art metaheuristics and forward-backward R&S

| **Algorithm** | **J30** | **J60** | **J120** |
|---|---|---|---|
| forward-backward R&S (our proposed method) | 0 | 10.53 | 31.07 |
| R&S (our proposed method) | 0 | 10.54 | 31.09 |
| GA(SA(FBI))[16] | 0.01 | 10.63 | 30.66 |
| GA(FBI)[23] | 0.02 | 10.68 | 30.82 |
| Sequential(SS(FBI))[32] | 0 | 10.58 | 31.16 |
| GA(FBI)[22] | 0.02 | 10.73 | 31.24 |
| GA(FBI)[20] | 0 | 10.57 | 31.28 |
| GA(FBI)[21] | 0 | 10.71 | 31.3 |
| PSO(LS) [59] | 0.05 | 10.62 | 31.43 |
| GA(FBI) [19] | 0.03 | 10.84 | 31.49 |
| ALNS(FBI)[60] | 0.02 | 10.73 | 31.54 |
| SS(EM + FBI)[61] | 0.01 | 10.71 | 31.57 |
| GA(FBI)[17] | 0.01 | 10.81 | 31.65 |
| PSO(FBI)[62] | 0.02 | 10.85 | 32.4 |
| EA(FBI)[25] | 0.03 | 10.91 | 32.52 |
| GA(FBI)[24] | – | 10.57 | 32.76 |
| PSO(FBI)[63] | 0.01 | 10.79 | 32.89 |
| GA(FBI)[18] | – | 10.66 | 33.82 |
| BCO(FBI)[28] | 0.04 | 11.16 | 34.55 |
| DBGA [15] | 0.02 | 10.68 | 30.69 |
| GA-FBI [64] | 0.0 | 10.56 | 32.76 |
| COA [65] | 0.0 | 10.58 | 31.22 |
| PSO-SS[66] | 0.01 | 10.68 | 31.23 |
| MA [7] | 0 | 10.55 | 31.12 |

used the CPLEX CP optimizer . Therefore, we compare the best results of the tested algorithms for 50,000 generated schedules, which time-wise is very close to the computational times of our methods. The average deviation from the critical path method (CPM), i.e., $\Delta_{CPM}$ is calculated by using Equation (6), where the lower bound of the makespan ($C_{max,LB,p}$) is obtained by CPM.

Table 2 for the instances with 30 tasks shows the average deviation of the solution of each algorithm from the optimal solutions. In this set of instances, forward-backward R&S can find the optimal solution for all the instances. The results for the other instances in this table are the average deviation of the solution for each algorithm from the CPM. In solving instances with 60 tasks, our method provides the lowest average deviation from the CPM among all the algorithms, so it is the best method for this set of instances. Dealing with the instances with 120 tasks, DBGA [15], GA(SA(FBI))[16], and GA(FBI)[23] reported better solution than our algorithm, and our method is better than the remaining state-of-the-arts

### 4.3 Discussion

Aiming to solve RCPSP efficiently, we proposed a R&S matheuristic that first generates an initial solution. Then, a rolling time window is employed to determine the tasks that are free to be reordered and those that their execution order is not allowed to be altered. The potential of improving each relaxed problem corresponds with the length of each time window, i.e., the wider the time window, the more tasks can be reordered to improve the results. However, increasing the time window length increases the problem's complexity. The parameter *overlap* between time windows allows reordering more tasks in two consecutive relaxed problems. Increasing this parameter is expected to result in increasing the computation time.

The main challenge with the structure of the proposed R&S algorithm is that the solution is highly impacted by the initial solution. Also, only the tasks in *overlap* can be passed to the next relaxed problem. The latter is more important for tasks with fewer predecessors and successors. In contrast with the tasks on a critical path, those tasks should be able to move in the time horizon easily to find the best starting time, leading therefore to improving the solution. To reduce the dependency of the algorithm on the initial solution and also to enhance the algorithm's exploration capability, we proposed to generate solutions in both forward and backward passes. Based on the presented results, employing the CPLEX CP optimizer within the R&S framework develops feasible solution for all tested instances. The CPLEX CP optimizer provides the optimum solution for instances with 30 tasks, and R&S is shown to improve the solutions obtained by CPLEX CP optimizer for the larger and challenging instances.

## 5 Conclusion

In this paper, a novel relax-and-solve matheuristic for the RCPSP was presented. In this algorithm, a solution is presented by a schedule that indicates the starting time of each task and then relaxed by removing precedence constraints for a set of tasks that are specified by a rolling time window. The relaxed problems are then solved by using the CPLEX CP optimizer in both forward and backward passes. A set of 1560 instances of PSPLIB with 30, 60 and 120 tasks were used to test the proposed algorithm. Instances with 30 tasks are easy to be optimally solved by the CPLEX CP optimizer and that quickly. However, for larger instances, i.e., instances with 60 and 100 tasks, our algorithm provides superior solutions in a shorter time to the stand-alone CPLEX CP optimizer. The results indicate that arming the R&S with forward and backward passed leads to better results than the R&S with only forward pass. Because generating a solution in both passes lets the algorithm do not trap in the local optima and reduces the dependency of the algorithm on initial solutions.

An important aspect towards the efficiency of our method lies in relaxing an instance without decreasing the number of tasks. In our method, certain

tasks are not allowed for their execution order to be altered. In this way, the obtained solution is always feasible.

The future research may focus on different schemes for relaxing the execution order of tasks. Also, extending the proposed method to solve multi-mode RCPSP and RCPSP under uncertainty is an interesting avenue for further research. Furthermore, the given structure of the R&S could be easily adapted for solving other optimization problems. Finally, the proposed R&S may benefit from available heuristics and metaheuristics, as well as exact solvers during the solve phase.

**Conflict of Interest:** The authors declare that they have no conflict of interest.

## References

[1] Herroelen, W., Leus, R.: Identification and illumination of popular misconceptions about project scheduling and time buffering in a resource-constrained environment. Journal of the Operational Research Society **56**(1), 102–109 (2005)

[2] Pritsker, A., Waiters, L.J., Wolfe, P.: Multiproject scheduling with limited resources: A zero-one programming approach. Management Science **16**, 93–108 (1969)

[3] Liu, J., Lu, M.: Optimization on Supply-constrained Module Assembly Process. In: IGLC 2017 - Proceedings of the 25th Annual Conference of the International Group for Lean Construction, pp. 813–820 (2017). https://doi.org/10.24928/2017/0104. cited By 2

[4] Alford, C., Brazil, M., Lee, D.H.: Optimisation in Underground Mining, pp. 561–577. Springer, Boston, MA (2007). https://doi.org/10.1007/978-0-387-71815-6_30. https://doi.org/10.1007/978-0-387-71815-6_30

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

[6] Blazewicz, J., Lenstra, J.K., Kan, A.H.G.R.: Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics **5**(1), 11–24 (1983). https://doi.org/10.1016/0166-218X(83)90012-4

[7] Rahman, H.F., Chakrabortty, R.K., Ryan, M.J.: Memetic algorithm for solving resource constrained project scheduling problems. Automation in Construction **111**, 103052 (2020). https://doi.org/10.1016/j.autcon.2019.103052

[8] Herroelen, W., De Reyck, B., Demeulemeester, E.: Resource-constrained

project scheduling: A survey of recent developments. Computers & Operations Research **25**(4), 279–302 (1998). https://doi.org/10.1016/S0305-0548(97)00055-5

[9] Chen, H., Ding, G., Qin, S., Zhang, J.: A hyper-heuristic based ensemble genetic programming approach for stochastic resource constrained project scheduling problem. Expert Systems with Applications **167**, 114174 (2021). https://doi.org/10.1016/j.eswa.2020.114174

[10] Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research **174**(1), 23–37 (2006). https://doi.org/10.1016/j.ejor.2005.01.065

[11] Li, K.Y., Willis, R.J.: An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research **56**(3), 370–379 (1992). https://doi.org/10.1016/0377-2217(92)90320-9

[12] Pellerin, R., Perrier, N., Berthaut, F.: A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. European Journal of Operational Research **280**(2), 395–416 (2020). https://doi.org/10.1016/j.ejor.2019.01.063

[13] Holland, J.H.: Genetic algorithms. Scientific American (1992)

[14] Agarwal, A., Colak, S., Erenguc, S.: A neurogenetic approach for the resource-constrained project scheduling problem. Computers & Operations Research **38**(1), 44–50 (2011). https://doi.org/10.1016/j.cor.2010.01.007. Project Management and Scheduling

[15] Debels, D., Vanhoucke, M.: A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. Operations Research **55**, 457–469 (2007). https://doi.org/10.1287/opre.1060.0358

[16] Lim, A., Ma, H., Rodrigues, B., Tan, S., Xiao, F.: New meta-heuristics for the resource-constrained project scheduling problem. Flexible Services and Manufacturing Journal **25**, 48–73 (2011). https://doi.org/10.1007/s10696-011-9133-0

[17] Cervantes, M., Lova, A., Tormos, P., Barber, F.: A dynamic population steady-state genetic algorithm for the resource-constrained project scheduling problem. In: Nguyen, N.T., Borzemski, L., Grzech, A., Ali, M. (eds.) New Frontiers in Applied Artificial Intelligence, pp. 611–620. Springer, Berlin, Heidelberg (2008)

[18] Ismail, I., Barghash, M.: Diversity guided genetic algorithm to solve the

resource constrained project scheduling problem. Int. J. of Planning and Scheduling **1**, 147–170 (2012). https://doi.org/10.1504/IJPS.2012.050125

[19] Alcaraz, J., Maroto, C., Ruiz, R.: Improving the performance of genetic algorithms for the rcps problem. Proceedings of the Ninth International Workshop on Project Management and Scheduling, 40–43 (2004). Cited By :39

[20] Wang, H., Li, T., Lin, D.: Efficient genetic algorithm for resource-constrained project scheduling problem. Transactions of Tianjin University **16**(5), 376–382 (2010). https://doi.org/10.1007/s12209-010-1495-y

[21] Zamani, R.: A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. European Journal of Operational Research **229**(2), 552–559 (2013). https://doi.org/10.1016/j.ejor.2013.03.005

[22] Valls, V., Ballestín, F., Quintanilla, S.: A hybrid genetic algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research **185**, 495–508 (2008). https://doi.org/10.1016/j.ejor.2006.12.033

[23] Debels, D., Vanhoucke, M.: A bi-population based genetic algorithm for the resource-constrained project scheduling problem. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganá, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) Computational Science and Its Applications – ICCSA 2005, pp. 378–387. Springer, Berlin, Heidelberg (2005)

[24] Fernando, G.J., C., R.M.G., M., M.J.J.: A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Journal of Heuristics **17**(5), 467–486 (2011). https://doi.org/10.1007/s10732-010-9142-2

[25] Chand, S., Singh, H.K., Ray, T.: A heuristic algorithm for solving resource constrained project scheduling problems. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 225–232 (2017). https://doi.org/10.1109/CEC.2017.7969317

[26] Deng, L., Lin, V., Chen, M.: Hybrid ant colony optimization for the resource-constrained project scheduling problem. Journal of Systems Engineering and Electronics **21**(1), 67–71 (2010). https://doi.org/10.3969/j.issn.1004-4132.2010.01.012

[27] Dorigo, M., Di Caro, G., Gambardella, L.M.: Ant algorithms for discrete optimization." artificial life 5, 137-172. Artificial Life **5**, 137–172 (1999). https://doi.org/10.1162/106454699568728

[28] Ziarati, K., Akbari, R., Zeighami, V.: On the performance of bee algorithms for resource-constrained project scheduling problem. Applied Soft Computing **11**(4), 3720–3733 (2011). https://doi.org/10.1016/j.asoc.2011.02.002

[29] Pham, D., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., Zaidi, M.: The bees algorithm technical note. Manufacturing Engineering Centre, Cardiff University, UK, 1–57 (2005)

[30] Karaboga, D.: An idea based on honey bee swarm for numerical optimization, technical report - tr06. Technical Report, Erciyes University (2005)

[31] A novel bee swarm optimization algorithm for numerical function optimization. Communications in Nonlinear Science and Numerical Simulation **15**(10), 3142–3155 (2010). https://doi.org/10.1016/j.cnsns.2009.11.003

[32] Berthaut, F., Pellerin, R., Hajji, A., Perrier, N.: A path relinking-based scatter search for the resource-constrained project scheduling problem. International Journal of Project Organisation and Management **10**(1), 1–36 (2018). https://doi.org/10.1504/IJPOM.2018.090372

[33] Gu, H., Schutt, A., Stuckey, P.J.: A lagrangian relaxation based forward-backward improvement heuristic for maximising the net present value of resource-constrained projects. In: Gomes, C., Sellmann, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 340–346. Springer, Berlin, Heidelberg (2013)

[34] Laborie, P.: An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling, pp. 403–411. Springer, Cham (2018)

[35] Maleck, C., Nieke, G., Bock, K., Pabst, D., Stehli, M.: A comparison of an cp and mip approach for scheduling jobs in production areas with time constraints and uncertainties. In: 2018 Winter Simulation Conference (WSC), pp. 3526–3537 (2018). https://doi.org/10.1109/WSC.2018.8632404

[36] Kelareva, E., Brand, S., Kilby, P., Thiebaux, S., Wallace, M.: Cp and mip methods for ship scheduling with time-varying draft. ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling (2012)

[37] Liess, O., Michelon, P.: A constraint programming approach for the resource-constrained project scheduling problem. Annals of Operations

Research **157**(1), 25–36 (2008)

[38] Schutt, A., Feydy, T., Stuckey, P., Wallace, M.: Explaining the cumulative propagator. Constraints **16**, 250–282 (2011). https://doi.org/10.1007/s10601-010-9103-2

[39] Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: A Satisfiability Solving Approach, pp. 135–160. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-05443-8_7. https://doi.org/10.1007/978-3-319-05443-8_7

[40] Kreter, S., Schutt, A., Stuckey, P.: Using constraint programming for solving rcpsp/max-cal. Constraints **22** (2017). https://doi.org/10.1007/s10601-016-9266-6

[41] Absi, N., van den Heuvel, W.: Worst case analysis of relax and fix heuristics for lot-sizing problems. European Journal of Operational Research **279**(2), 449–458 (2019). https://doi.org/10.1016/j.ejor.2019.06.010

[42] Helber, S., Sahling, F.: A fix-and-optimize approach for the multi-level capacitated lot sizing problem. International Journal of Production Economics **123**(2), 247–256 (2010)

[43] Escudero, L.F., Romero, C.P.: On solving a large-scale problem on facility location and customer assignment with interaction costs along a time horizon. TOP: An Official Journal of the Spanish Society of Statistics and Operations Research **25**(3), 601–622 (2017). https://doi.org/10.1007/s11750-017-0461-4

[44] Salehipour, A.: A heuristic algorithm for the aircraft landing problem. In: 22nd International Congress on Modelling and Simulation (2017). Modelling and Simulation Society of Australia and New Zealand Inc.(MSSANZ)

[45] Salehipour, A., Ahmadian, M., Oron, D.: Efficient and simple heuristics for the aircraft landing problem. In: Matheuristic 2018 International Conference (2018)

[46] Ahmadian, M.M., Salehipour, A., Kovalyov, M.: An efficient relax-and-solve heuristic for open-shop scheduling problem to minimize total weighted earliness-tardiness. Available at SSRN 3601396 (2020)

[47] Ahmadian, M.M., Salehipour, A., Cheng, T.C.E.: A meta-heuristic to solve the just-in-time job-shop scheduling problem. European Journal of Operational Research **288**(1), 14–29 (2021)

[48] Etminaniesfahani., A., Gu., H., Salehipour., A.: An Efficient Relax-and-Solve Algorithm for the Resource-Constrained Project Scheduling Problem. In: Proceedings of the 11th International Conference on Operations Research and Enterprise Systems - ICORES,, pp. 271–277. SciTePress. https://doi.org/10.5220/0010772400003117. INSTICC

[49] Gu, H., Schutt, A., Stuckey, P.J., Wallace, M.G., Chu, G.: In: Schwindt, C., Zimmermann, J. (eds.) Exact and Heuristic Methods for the Resource-Constrained Net Present Value Problem, pp. 299–318. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-05443-8_14. https://doi.org/10.1007/978-3-319-05443-8_14

[50] Laurent Perron and Vincent Furnon: OR-Tools version 7.2 (2019-7-19). https://developers.google.com/optimization/

[51] CPLEX, I.I.: version 12.8.0. IBM Corp., Armonk, New York, U.S. (2017)

[52] M. Pour, S., Drake, J.H., Ejlertsen, L.S., Rasmussen, K.M., Burke, E.K.: A hybrid constraint programming/mixed integer programming framework for the preventive signaling maintenance crew scheduling problem. European Journal of Operational Research **269**(1), 341–352 (2018). https://doi.org/10.1016/j.ejor.2017.08.033

[53] Bockmayr, A., Hooker, J.N.: Constraint programming. In: Aardal, K., Nemhauser, G.L., Weismantel, R. (eds.) Discrete Optimization. Handbooks in Operations Research and Management Science, vol. 12, pp. 559–600. Elsevier. https://doi.org/10.1016/S0927-0507(05)12010-6. https://www.sciencedirect.com/science/article/pii/S0927050705120106

[54] Kolisch, R., Hartmann, S.: Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, pp. 147–178. Springer, Boston, MA (1999). https://doi.org/10.1007/978-1-4615-5533-9_7. https://doi.org/10.1007/978-1-4615-5533-9_7

[55] Kolisch, R., Sprecher, A.: Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program. European Journal of Operational Research **96**(1), 205–216 (1997). https://doi.org/10.1016/S0377-2217(96)00170-1

[56] Vilím, P., Laborie, P., Shaw, P.: Failure-directed search for constraint-based scheduling. In: Michel, L. (ed.) Integration of AI and OR Techniques in Constraint Programming, pp. 437–453. Springer, Cham (2015)

[57] Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent, I.P. (ed.) Principles and Practice of Constraint Programming - CP 2009, pp. 352–366. Springer, Berlin, Heidelberg (2009)

[58] Bofill, M., Coll, J., Suy, J., Villaret, M.: Smt encodings for resource-constrained project scheduling problems. Computers & Industrial Engineering **149**, 106777 (2020). https://doi.org/10.1016/j.cie.2020.106777

[59] Czogalla, J., Fink, A.: Particle Swarm Topologies for Resource Constrained Project Scheduling, pp. 61–73. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03211-0_6. https://doi.org/10.1007/978-3-642-03211-0_6

[60] Muller, L.F.: An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In: Proceedings of the VIII Metaheuristics International Conference (MIC) (2009)

[61] Debels, D., De Reyck, B., Leus, R., Vanhoucke, M.: A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. European Journal of Operational Research **169**(2), 638–653 (2006). https://doi.org/10.1016/j.ejor.2004.08.020. Feature Cluster on Scatter Search Methods for Optimization

[62] Fahmy, A., Hassan, T.M., Bassioni, H.: Improving rcpsp solutions quality with stacking justification – application with particle swarm optimization. Expert Systems with Applications **41**(13), 5870–5881 (2014). https://doi.org/10.1016/j.eswa.2014.03.027

[63] Nasiri, M.M.: A pseudo particle swarm optimization for the rcpsp. The International Journal of Advanced Manufacturing Technology **65**(5), 909–918 (2013). https://doi.org/10.1007/s00170-012-4227-8

[64] Liu, J., Liu, Y., Shi, Y., Li, J.: Solving resource-constrained project scheduling problem via genetic algorithm. Journal of Computing in Civil Engineering **34**(2), 04019055 (2020). https://doi.org/10.1061/(ASCE)CP.1943-5487.0000874

[65] Elsayed, S., Sarker, R., Ray, T., Coello, C.C.: Consolidated optimization algorithm for resource-constrained project scheduling problems. Information Sciences **418-419**, 346–362 (2017). https://doi.org/10.1016/j.ins.2017.08.023

[66] Koulinas, G., Kotsikas, L., Anagnostopoulos, K.: A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. Information Sciences **277**, 680–693 (2014). https://doi.org/10.1016/j.ins.2014.02.155