IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 30, 2021, accepted September 12, 2021, date of publication September 22, 2021,
date of current version September 30, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3114702

Genetic Algorithm for Singular Resource Constrained Project Scheduling Problems

FIROZ MAHMUD, FORHAD ZAMAN, ALI AHRARI, RUHUL SARKER, (Member, IEEE), AND DARYL ESSAM, (Member, IEEE)

School of Engineering and Information Technology, University of New South Wales Canberra, Canberra, ACT 2612, Australia

Corresponding author: Firoz Mahmud (firoz.mahmud@student.adfa.edu.au)

The work of Ruhul Sarker and Daryl Essam was supported in part by the Australian Research Council Discovery Projects under Grant DP210102939.

ABSTRACT The Resource-Constrained Project Scheduling Problem (RCPSP) is a challenging optimization problem. In RCPSPs, it is very common to consider homogeneous activities, which means all activities require all types of resources. In practice, the activities are often singular because they usually require one single resource to execute an activity. The existing algorithms may be used for solving this variant of RCPSPs with a simple modification. However, they are computationally expensive due to unnecessary resource constraints. In this paper, we propose a customised evolutionary algorithm integrated with three heuristics for the singular activities. The first heuristic is based on the earliest start time with an aim to rectify an infeasible schedule. The second heuristic is based on neighbourhood swapping which is used to find the best possible alternatives. The third heuristic is used to further enhance the quality of the schedule. The performance of the proposed framework has been tested by solving a wide range of benchmark problems and the obtained results revealed that the proposed approach outperformed the existing algorithms. In addition, statistical and parametric testing show the value and characteristics of the proposed approach.

INDEX TERMS Resource-Constrained Project Scheduling, singular activities, genetic algorithm, neighbourhood swapping, forward-backward improvement.

I. INTRODUCTION

A project is a collection of activities where each activity must execute once to accomplish a project. The efficiency of a project depends on managing the order of execution of the activities where many interactions may exist among the activities, such as precedence relationships among the activities and sharing a resource by multiple activities. In order to execute an activity, one or more resources are required for a finite duration; however, available resources are usually limited. In order to find the best schedule, the project manager must assign the resources appropriately. The scheduling of the activities, which is recognized as project scheduling, is a well-known complex optimization problem. In this context, the activity scheduling with resource allocation is called the Resource-Constrained Project Scheduling Problem (RCPSP).

Over time, RCPSPs have attracted the attention of many researchers and practitioners due to the practicability of the research in this field to real-world problems, such as manufacturing industry, construction, aircraft, engineering, and software [1]. As RCPSP is an NP-hard (computational complexity) problem [2], [3], many methods have been introduced to solve a wide variety of RCPSPs problems in which the objective is to minimize the makespan while satisfying precedence and resource constraints such as mathematical model based exact solution, heuristic-based algorithm, meta-heuristic algorithm, and hybrid meta-heuristic based approaches [4]. Exact approach can find the global optimum in low-dimensional problems quickly, but they become computationally expensive and inefficient for large-scale problems. To overcome this limitation, many heuristic and meta-heuristic algorithms have been developed, such as genetic algorithm (GA) [5]–[7], hybrid GA (HGA) [8], decomposition-based GA [9], differential evolution (DE) [5], [10], variable neighbourhood search heuristic (MVNSH) [11]. A detailed literature review of heuristic and meta-heuristic approaches is discussed in section II, where the following research gaps are elucidated. (i) No single

The associate editor coordinating the review of this manuscript and approving it for publication was Hisao Ishibuchi.

algorithm can find quality solution for a wide range of RCPSPs. One algorithm may perform better for some problems that may perform badly for some other problems, or some algorithms may be suitable for small scale RCPSPs, however they are computationally expensive for large-scale problems. (ii) It is seen that for some RCPSPs, the best solutions found by the state-of-the-art methods are still far from the best-known solutions for those problems.

The above-mentioned solution approaches for solving different RCPSPs have been developed with a common assumption that activities require multiple different types of resources. However, many real-world problems are singular, meaning each activity require exactly one type of renewable resource to complete its operation. Fig. 1 shows a typical software or app (mobile application) development process cycle as an example of singular activities. In the software industry, different types of skills are needed to develop software such as: (a) Software designer for designing the layout of the software, (b) Programmer to convert the layout into executable software, and (c) A testing person to test/debug the software according to the requirements.

This example has three different tasks and each task requires a person with a different skill. It is clear that one task cannot start before finishing of the earlier task. For example, a programmer cannot start coding before getting the layout. In traditional problem definition, all of these people are assumed to be involved in executing each and every activity, which is not the practical scenario and thus has unnecessary resource requirements assumption. This assumption may significantly increase the complexity of the problem that would lead to poor performance as well as an increase in the computational budget. To deal with such real-world problems, in this paper, we define the problems by assuming that each activity requires only one type of resource. For example, software designer is required to design the layout of a software, in that case we only use the designer as the resource for software designing activity while other resource requirements (programmer and tester) are set as zero.

In this research, we introduced a class of RCPSPs where a project may require multiple resources but each activity needs only one of those resources for its completion. This variant of RCPSPs is also NP hard but the algorithmic computational complexity for this case is lower than the conventional RCPSPs. However, it is a challenging variant. From the best of our knowledge, no solution approach reported in the literature for such a variant of RCPSP. To solve such RCPSPs, in this paper, we propose an evolutionary algorithm based framework that combines a multi-operator genetic algorithm with three heuristics. In this approach, the initial population is randomly generated that may include some infeasible solutions. The first heuristic is used to repair an infeasible schedule (if any). Each of the individuals can usually be improved by the second heuristic where a swap-eligible matrix is introduced based on the direct and/or indirect successors and the singular resource requirements. We only swap those activities that require the same types

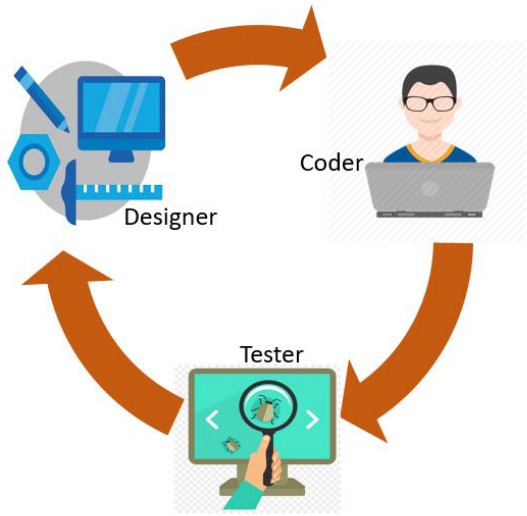


FIGURE 1. An example of a singular activities.

of resources because the makespan of the project depends on those activities. It is worth to mention here that a quality initial population helps the evolution process to converge to high quality solutions with less computational time. Finally, we use a forward-backward improvement (FBI) heuristic as a local search, for fine tuning the final solution for any possible improvement. To show the applicability of the proposed approach and analyze its performance, we proposed a new set of test problems using existing homogeneous RCPSPs taken from the standard project scheduling library (PSPLIB [12]). Here, we developed a systematic approach to generate singular RCPSPs, as discussed in subsection V-A. The performance of the proposed framework was evaluated by solving these test problems, with up to 120 non-dummy activities. To judge the quality of solutions, the experimental results obtained from the proposed approach were compared with the same from a well-known algorithm, known as Consolidated Optimization Algorithm (COA), which was developed for homogeneous RCPSPs. The comparison shows the superiority of the proposed framework.

The remainder of this paper is organized as follows: a literature review is presented in Section II. Section III gives the problem description. The details of the proposed framework is presented in Section IV. Section V provides the experimental result and performance comparison. Section VI presents the conclusion and recommendations for future work.

II. LITERATURE REVIEW

Over the last few decades, many solution approaches have been developed for solving different RCPSP variants, which can be classified as (i) exact approach, (ii) heuristic and meta-heuristic algorithms, and (iii) hybrid algorithm. For the exact approach, mathematical modelling based approaches such as integer programming [13], [14] and mixed integer linear programming [15], [16] are widely used. The branch and bound (BB) [17], cutting plane (CP) [18], and branch and cut (BC) [19], [20] algorithms are usually applied to

solve the developed models. Although the exact method can solve a small-scale problem effectively, it's performance deteriorates and they become computationally expensive for large-scale problems [21]. For solving medium to large scale problems, heuristic and meta-heuristics are used, such as priority rule-based heuristic [22], [23], GA based solution [24]–[26], GA with implicit enumeration module (IEM) based framework [27], combined precedence relations and resource calendars based approach [28], fuzzy clustering-based chaotic technique with a differential evolution (FCDE) [29], particle swarm optimisation (PSO) [30], simulated annealing [31], and so on.

Although heuristic and meta-heuristic methods are very effective to solve a wide variety of RCPSPs, it cannot guarantee to find a satisfactory solution for all RCPSP variants. To overcome this limitation, hybrid algorithms that combine heuristics and meta-heuristics were developed. For example, Elsayed *et al.* [32] proposed a hybrid framework for solving single-mode RCPSP using two multi-operator evolutionary algorithms i.e., multi-operator GA (MOGA) and multi-operator DE (MODE), while Zaman *et al.* [33] solved multi-mode RCPSPs based on this framework. Multi-mode RCPSPs are an extension of classical single-mode RCPSPs where each activity has several execution modes and each mode has a different set of duration and resource requirements. Both algorithms have their own sub-populations and new offspring are generated by evolving its own individual sub-population while considering self-adaptive crossover and mutation operators. A final set of solutions, which is obtained from both algorithms, is further enhanced by using a local search, namely forward-backward improvement. The performance of the framework is evaluated by solving standard benchmark problems and shows promising results; however, it was not tested on 90 non-dummy activities benchmark problems. Bettemir *et al.* [31] proposed an optimization model using a combination of two meta-heuristic algorithms i.e., genetic algorithm and simulated annealing for achieving faster convergence rate and increasing efficiency. For comparison, seven heuristics were included which were taken from different algorithms used in project management. The experimental result showed that the proposed hybrid algorithm outperforms a sole GA based meta-heuristic, however, it is outperformed by recent algorithms. Fang *et al.* [34] proposed a framework for solving RCPSP using two different heuristic algorithms (memetic GA and PSO). Although the inclusion of a local search improves the quality of solutions, it was inferior for 30 activities problems. Ziarati *et al.* [35] developed a framework by using three variants of bee algorithms i.e., bee algorithm (BA), bee swarm optimization (BSO), and artificial bee colony (ABC) algorithm. The experimental result showed that BA with FBI outperformed in comparison with two other variants. However, it was far away from the state-of-the art algorithms.

All the above solution approaches were developed for solving homogeneous activities, whereas, singular activities are more practical for the real-world problem as introduced in section I. To the best of our knowledge, RCPSP with singular activities and its solution approach are applicable for a wide variety of benchmark problems and has not yet been explored much except for [4], [36], where two random problem instances from each of the problem instances from standard datasets(i.e., j30, j60, j90, and j120) were considered.

III. PROBLEM DESCRIPTION

The RCPSP can be formulated as a mathematical model, with an objective of minimizing the project's duration (makespan) while satisfying the relevant constraints such as precedence and resource constraints. A project S consists of $D+2$ activities and can be defined as: $\mathbf{S} = \{1, 2, \ldots, D+2\}$, where each activity has to execute exactly once in order to complete a project. Two dummy activities: 1 and $D+2$ are used to represent the 'project start' and 'project finish', respectively. To execute an activity, a certain amount of time and renewable resources are required. We use activity on node (AoN) paradigm to represent a project instead of activity on arc (AoA) diagram. It is easier to create a base model of a project and more efficient to manage the logical relation between the activities by using AoN rather than AOA [5]. For example, Fig. 3 (a) is an AoN representation of an example project with 7 activities ($S = \{1, 2, \ldots, 7\}$), where arc represent the precedence constraints, which enforce that an activity cannot start before finishing the predecessor(s) of that activity. The acronyms used to formulate the model and proposed approach are presented in Table 1.

To formulate the mathematical model, in this research, we consider the following assumptions: (1) each activity can only execute once; (2) an activity cannot start execution before finishing all of its predecessors; (3) duration and resource requirement of each activity is given in advance; (4) resource requirement by an activity must not exceed the maximum available resource limit and can only use one renewable resource type; (5) Preemption is not allowed, that means an activity cannot be interrupted in the middle of the execution; (6) the objective of this optimization model is to minimize the project completion time. The mathematical model is as follows [37]:

$$\text{Min: } FT_{D+2} \tag{1}$$

$$\text{Subject to: } FT_h \leq FT_j - d_j, \quad \forall h \in P_j, \quad \forall (j,h) \in \{1, 2, \ldots, D+2\} \tag{2}$$

$$\sum_{j \in CS_t} r_{j,k} \leq R_k, \ \forall k \in \{1, 2, \ldots, K\}, \forall CS_t \tag{3}$$

$$FT_j \geq 0, \quad \forall j \in \{1, 2, \ldots, D+2\} \tag{4}$$

where FT_{D+2} is the finish time (FT) of the last dummy activity, P_j is the list of all predecessors for the j^{th} activity, $r_{j,k}$ is the amount of the k^{th} resource required for the j^{th} activity, K is the total number of renewable resource types, R_k is the resource availability of the k^{th} renewable resource type and CS_t is the set of activities scheduled at time $t \leq FT_{D+2}$. The objective function of this model is presented by using Eqn. (1), which is the makespan minimization of a project.

TABLE 1. List of acronyms.

RCPSP Model	
D	Number of non-dummy activities of a project.
j	An activity in a project, i.e., $j = 1, 2, ..., D + 2$.
FT_j	Finish time of j^{th} activity.
F_{D+2}	Finish time of last dummy activity.
d_j	Duration of j^{th} activity.
P_j	A set of immediate predecessor activities of j^{th} activity.
$r_{j,k}$	Ammount of k^{th} renewable resource requirement by j^{th} activity.
R_K	Maximum amount of resource availability of k^{th} renewable resources.
K	Total number of renewable resources.
CS	A set of all non-dummy activities of a project, i.e., $j = 2, 3, ...D + 1$.
CS_t	Set of activities running at t^{th} time period, i.e., $CS_t \in CS$.
t	Discrete time period
Algorithm	
$MRuns$	Maximum number of runs.
CFE	Current fitness function evaluation.
$MFFE$	Maximum number of fitness function evaluations.
N_P	Population size.
i	An individual from population set, i.e., $i \in N_p$.
X_i	Acivities sequence of i^{th} individual.
EST_j	Earliest start time of j^{th} activity
FT_j	Finish time of j^{th} activity
SEM	Swap eligible matrix.
$NoAct$	Total number of activities in an individual.
row	Current row number.
col	Current column number.

Precedence and resource constraints are shown in Eqns. (2) and (3), respectively. Eqn. (4) certifies that the FT of each activity must be greater than zero or equal to zero.

IV. PROPOSED APPROACH

In this subsection, we discuss our proposed evolutionary algorithm for solving large-scale RCPSP problems. From the many evolutionary algorithms, we chose multi-operator genetic algorithm due to its superior performance to solve complex real-world problems [32]. In our experimental setup, we modified the original standard benchmarks according to the requirement of singular RCPSPs, where an activity requires one renewable resource instead of all available renewable resources. We named the proposed algorithm as S-GA. The proposed S-GA and the three heuristics i.e., earliest start time (EST), neighbourhood swapping (NS), and forward-backward improvement based heuristic are described in the following subsections. The details of the proposed approach is shown in Fig 2.

A. S-GA FRAMEWORK

The proposed framework starts with a random initial population $\mathbf{X_i}$, $i = 1, 2, \ldots, N_p$, where N_p is the population size. Some of the initial solutions may be infeasible with respect to precedence and resource constraints. To repair an infeasible schedule to a feasible one, an EST based heuristic is used which will be discussed in subsection IV-C. The EST and FT of each activity are also calculated using Eqns. (6) and (7), respectively. The feasible schedule is further verified with the resource constraints. After satisfying those two constraints, the fitness value (FV) of each population is calculated as FT_{D+2}. The feasible $\mathbf{X_i}$ is the initial population for heuristic 2. This heuristic defines a swap eligible matrix to determine the eligible activities for swapping, which will be explained in subsection IV-D.

All the resultant solutions are then sorted based on their FVs. To obtain a more diverse population, a new set of solutions is generated based on genetic operators, including two crossover operators and one left-shift mutation operator. which will be elaborated in subsection IV-E. These new solutions are then repaired using our EST heuristic. Once we obtain the new feasible individual from GA, the redundant individual (if any) is removed from $\mathbf{X_i}$ and new random individual is inserted to replace each of the redundant individuals. In the next step, the quality of $\mathbf{X_i}$ is further enhanced by using FBI, which will be discussed in subsection IV-F. The process of sampling new solutions by the GA, applying the heuristics to each individual, and updating $\mathbf{X_i}$ continues until the maximum number of fitness evaluations is reached. The detailed steps of the proposed S-GA is shown in Algorithm 1.

B. REPRESENTATION AND INITIAL POPULATION

In this subsection, we represent the initial population with size N_p as $\mathbf{X_i}$. Each schedule is a random sequence of the activities with the size of $D+2$. The random initial population is generated as follows:

$$X_i = \{1, perm(\Delta)_i, D + 2\}, \quad \forall i, \Delta = \{2, 3, \ldots, D + 1\}. \tag{5}$$

where $perm(\Delta)_i$ is the random combination of non-dummy activities, 1 and $D + 2$ represent two dummy activities for 'start' and 'end' activities, respectively, $\mathbf{X_i}$ is the i^{th} schedule, $i \in N_p$.

C. HEURISTIC 1: REPAIRING AN INFEASIBLE SOLUTION

In this research, the initial schedule $\mathbf{X_i}$ is a permutation or random combination of non-dummy activities. This random schedule may be infeasible based on precedence and resource constraints. A serial schedule generation scheme (SSGS) [38] is used to create a feasible schedule from the infeasible one. In SSGS, one activity is selected in each iteration based on earliest start time from the non-dummy activities by satisfying precedence and resource constraints. This process is continue until all non-dummy activities are converted to feasible schedule. In order to make all activities of i^{th} individual are feasible, D iterations are required [39]. Fig. 3 (a) shows an example of a project network, Fig. 3 (b) shows a feasible schedule that only considers precedence constraints and the makespan is 7 units, and Fig. 3 (c) shows a feasible schedule

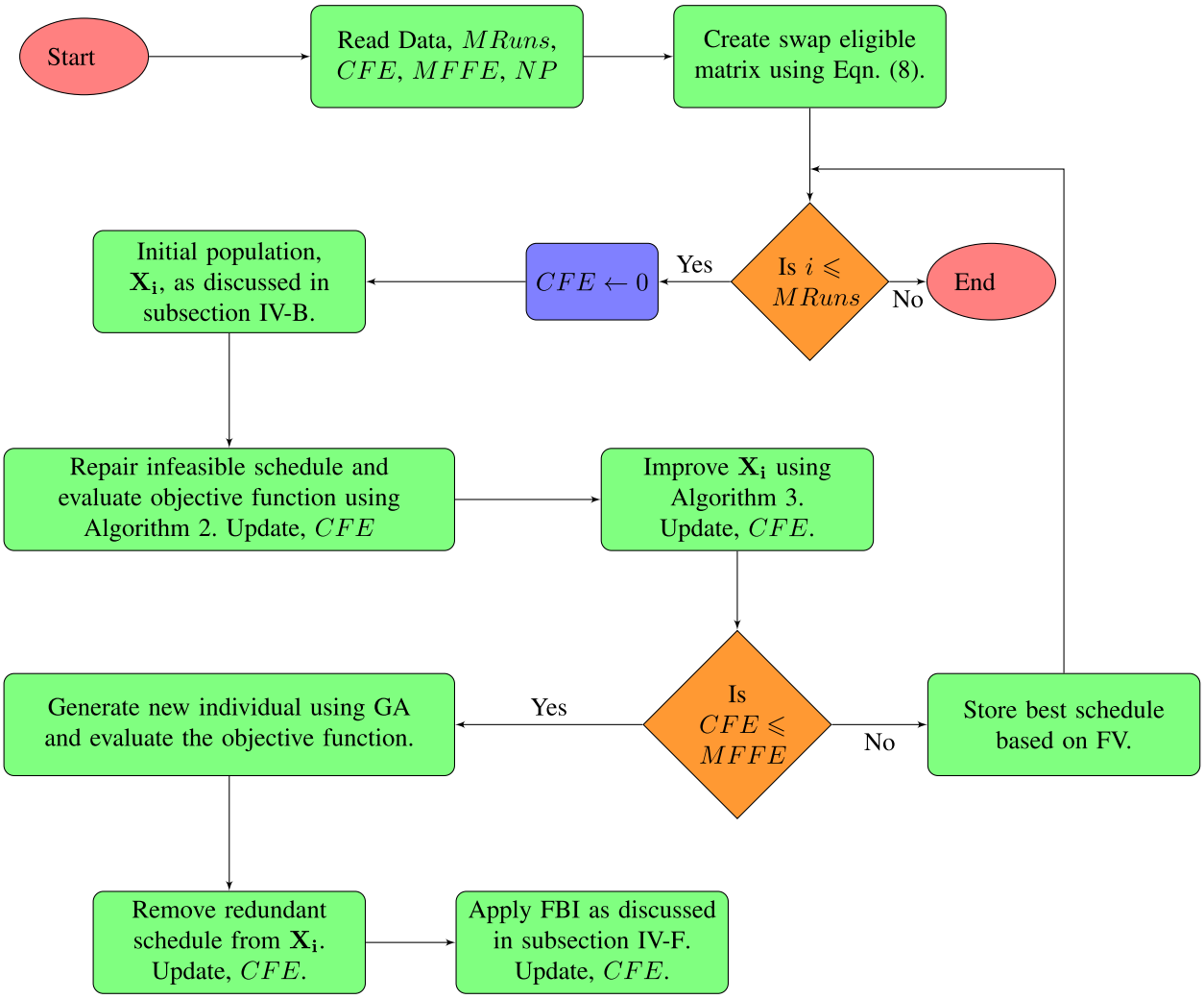


FIGURE 2. Flowchart of Algorithm 1.

after satisfying both precedence and resource constraints where the makespan is 10 units instead of 7. The following steps are taken into account to construct feasible schedule.

Firstly, any infeasible schedule in $\mathbf{X_i}$ is repaired by using Eqn. (2), then EST_j and FT_j of the j^{th} activity are calculated using Eqns. (6) and (7), respectively by using the critical path method (CPM), while considering precedence constraints only.

$$EST_j = \begin{cases} 0, & \text{if } j \text{ has no predecessor} \\ max\big(FT_m | m \in P(j)\big), & 0 < j \leq D+2 \end{cases} \tag{6}$$

$$FT_j = EST_j + d_j \tag{7}$$

where FT_m is the finish time of the m^{th} predecessor from the set of predecessors of the j^{th} activity, j is the j^{th} activity, and $D+2$ represents the total number of activities.

Secondly, after converting to be precedence feasible $\mathbf{X_i}$, this schedule may violate the resource constraints. If any activity resource usages exceed the resource availability, it is rescheduled to the next earliest possible start time. Finally, makespan is calculated as the finish time of the last dummy activity, i.e., FT_{D+2}.

D. HEURISTIC 2: IMPROVEMENT OF $\mathbf{X_i}$, $i \in N_p$: NEIGHBORHOOD SWAPPING

In this subsection, NS is proposed to improve the quality of $\mathbf{X_i}$ which is obtained from Heuristic 1 as discussed in subsection IV-C. In NS, the activities of $\mathbf{X_i}$ are reshuffled between the eligible activities. The eligible activities refers to the activities which are not direct and/or indirect successors of each other but have the same type of resource requirements. The performance of a schedule depends on the optimal resource allocation among the activities. As we consider singular RCPSPs, where each activity requires one resource type, the performance may improve by swapping the activities that have the same type of resource requirement. As a consequence, we consider those activities for swapping, which are not direct and/or indirect successor, as well as have the same type of resource requirement.

The simplest way to rearrange the activities is a random combination. However, that may violate the precedence and

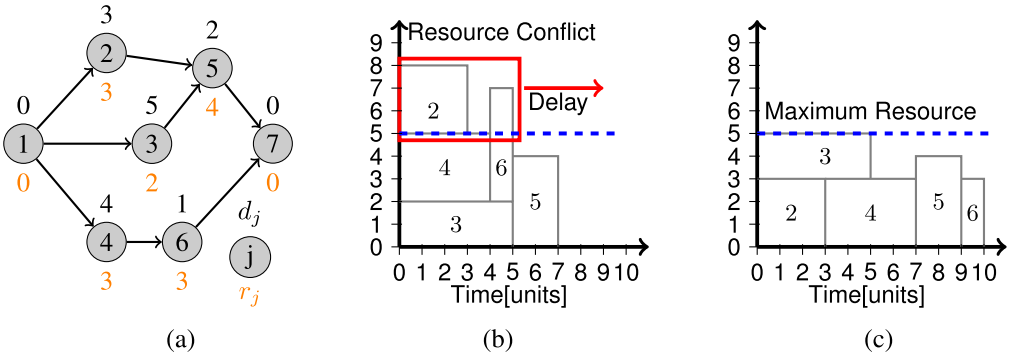


FIGURE 3. An example of (a) A project network, (b) A precedent feasible schedule, and (c) A resource feasible schedule.

Algorithm 1 Pseudo-Code of Proposed Solution Approach

Require: $MRuns$, CFE, $MFFE$, N_P.

1: Generate swap eligible matrix, as discussed in subsection IV-D.
2: **for** i = 1: MRuns **do**
3: $CFE \leftarrow 0$
4: **Initial Population:** Generate random initial schedule, $\mathbf{X_i}$, $i = 1, 2, \ldots, N_P$, as discussed in subsection IV-B.
5: Repair any infeasible schedule of $\mathbf{X_i}$, $i \in N_P$ by using Algorithm 2.
6: Evaluate objective function using Eqn. (1) . Update, $CFE \leftarrow CFE + N_P$.
7: Regenerate and improve $\mathbf{X_i}$, $i \in N_P$ using Algorithm 3. Increment CFE by 1 for each new schedule .
8: **while** $CFE \leqslant MFFE$ **do**
9: Generate offspring using the GA operators as discussed in subsection IV-E.
10: Evaluate objective function of the offspring based on steps 5 and 6.
11: Remove redundant schedule, if any in $\mathbf{X_i}$ and update, CFE=CFE+1 for each new schedule.
12: Apply forward-backward improvement (FBI) as discussed in subsection IV-F.
13: **end while**
14: Calculate the best individual.
15: **end for**

Algorithm 2 EST Technique to Obtain Feasible $\mathbf{X_i}$

Require: Number of activities, $D + 2$.

1: Generate a random initial schedule, $\mathbf{X_i}$, $i \in D + 2$
2: Repair infeasible schedule, if $\mathbf{X_i}$ violate the precedence constraints by using Eqn. (2).
3: Calculate EST_j and FT_j using Eqn. (6) and (7).
4: Repair infeasible schedule with respect to resource constraints. If any activity of $\mathbf{X_i}$ is violate then shift this activity to next possible start time.
5: Evaluate fitness value of each schedule.

resource constraints which leads to an infeasible schedule. For example, Fig. 4 (a) represent a feasible schedule after satisfying precedence and resource constraints of a project network (Fig. 3 (a)), Fig. 4 (b) shows a schedule after performing random shuffle which is infeasible as it interchanges the positions of activity-2 and activity-5. This schedule is infeasible as activity-2 is the immediate predecessor of activity-5, so activity-5 cannot be scheduled before activity-2. For the same reason, we cannot interchange activity-4 and activity-6. To avoid this unexpected event, we use the NS technique. As shown in Fig. 4 (c), we cannot swap those activities if they are a direct and indirect successor of each other. In Fig. 4 (c), the activities in red colour circles represent an unsuccessful swap and the green colour circle shows the successful swap as there is no direct or indirect successor relationship and the last one is a feasible schedule. To represent the successor relationships among the activities, we create a swap eligible matrix [4] by using Eqn. (8).

$$SEM_{q,j} = \begin{cases} 0, & \text{if } q == j \text{ and } r_{q,k} \neq r_{j,k} \\ 0, & \text{if } q > j,\ j \in P_q \text{ and } r_{q,k} \neq r_{j,k} \\ 0, & \text{if } j > q,\ q \in P_j \text{ and } r_{q,k} \neq r_{j,k} \\ 1, & \text{otherwise} \end{cases} \tag{8}$$

where $r_{j,k}$ and $r_{q,k}$ are the k^{th} types of resource demands of the j^{th} and q^{th} activity, respectively, P_j and P_q represents the predecessors of the j^{th} and q^{th} activity, respectively.

After creating the swap eligible matrix based on the precedence relation and the resource demand, this heuristic is applied to improve the quality of a feasible schedule, as shown in Algorithm 3. In it, two activities interchange their position where the cell value of the matrix is '1' and the makespan of the new schedule is evaluated. If the new makespan is better than the old makespan, then the original schedule is replaced by the new schedule, otherwise, $\mathbf{X_i}$ is unchanged.

E. GA BASED SCHEDULING

In this subsection, we discuss the GA and its parameters. GA is a powerful evolutionary algorithm proposed by Holland [40], to find good quality solutions using natural

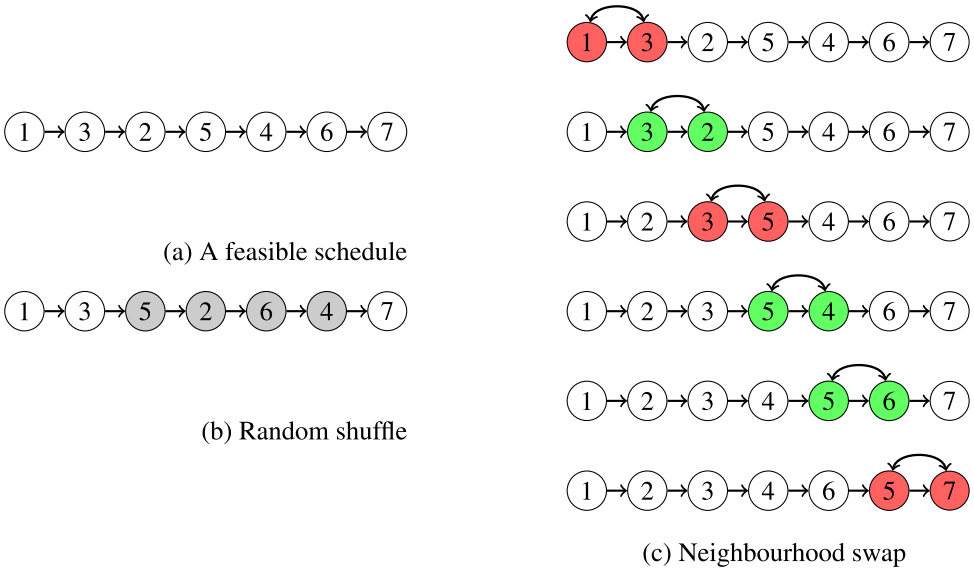


FIGURE 4. An example of neighbourhood swapping technique to improve $\mathbf{X_i}$ of Fig. 3(a).

Algorithm 3 Neighbourhood Swapping Based Scheduling

Require: A feasible $\mathbf{X_i}$, and its makespan, N_p, SEM.

```
for i = 1: N_p do
    for row = 2: NoAct-1 do
        for col = 2: NoAct-1 do
            Generate new schedule by interchanging position
            of two activities where cell value of the SEM
            matrix is '1'.
            Evaluate makespan_new of the new X_i, using Eqn.
            (1).
            if makespan_new < makespan then
                Update X_i.
            end if
        end for
    end for
end for
Improved X_i
```

selection and genetics. GA can produce a large number of diverse solution spaces from a limited number of random initial populations by using genetic operators, such as crossover, selection, and mutation. Since the target of each evolution is to find a better individual than the predecessor, GA keeps the better individuals, thus leading to achieving the goal of optimising the objective function.

In this research, the initial population ($\mathbf{X_i}$, $i \in N_p$) is randomly generated as discussed in subsection IV-B. To produce the offspring from $\mathbf{X_i}$, we use two crossovers: (1) two-point, and (2) uniform crossover and one mutation operator [32]. At the beginning, both crossover operators have an equal probability to select and produce the new offspring. Hence initially, the probability of each operator is set to 0.5. Based on the fitness value of a new individual, the probability of each crossover is adapted. If a new individual generated from a specific crossover operator is better than its parent, then the probability of that crossover operator is increased. As a result, more individuals are generated by using the more successful crossover operator. To create more diversity in $\mathbf{X_i}$, a mutation operator, namely left-shift mutation operator is used. At the initial point the mutation probability, MP, is set to 0.05, which is reduced to 0.001 at the later evolutionary process. The fitness value of the new individuals i.e., offspring are evaluated and the redundant offspring are removed by inserting a random schedule. Finally, individuals are sorted based on the best fitness value i.e., makespan.

F. HEURISTIC 3: FORWARD-BACKWARD IMPROVEMENT

In this research, the quality of $\mathbf{X_i}$ is further improved by FBI. In this method, also called double justification [41], firstly an activity is scheduled in a forward direction in one cycle, then in the next cycle, the same procedure is performed in a backward direction. The detailed steps of this approach can be found in [42].

In a forward cycle, activities are scheduled based on EST as discussed in subsection IV-C. In a backward cycle, FT_{D+2} which is obtained by a forward cycle is the start time of the next backward schedule. Each activity is shifted to its right as much as possible while maintaining precedence and resource constraints. $\mathbf{X_i}$, is updated if the makespan of a schedule is better than the original schedule.

V. EXPERIMENTAL RESULTS

This section presents the experimental setup, discusses and analyzes the results obtained by the proposed approach for solving the modified RCPSPs, including problem instances with 30, 60, 90, and 120 non-dummy activities (j30, j60, j90, and j120, respectively). The problem sets j30, j60, j90 and

j120 have 480, 480, 480, and 600 instances, respectively. To evaluate the performance of the proposed algorithm, all the $3 \times 480 + 600 = 2040$ problem instances were solved using five variants of the proposed algorithm as:

- **var1:** In this variant, a schedule is randomly generated which may be infeasible. The EST-based heuristic is used to convert any infeasible schedule to a feasible schedule by satisfying precedence constraint. The fitness value of the feasible schedule is evaluated after satisfying the resource constraints. This variant uses steps 2 to 6 of Algorithm 1.
- **var2:** The random initial schedule from var1 is further improved by using neighbourhood swapping (NS) as discussed in subsection IV-D. For this variant, we use steps 2 to 7 of Algorithm 1.
- **var3:** To create a more diverse solution as well as to improve the quality of the solution, a GA based heuristic is used as discussed in subsection IV-E. Steps 2 to 10 are used to evaluate the objective function.
- **var4:** Instead of using a random initial population in GA, in this variant, $\mathbf{X_i}$ is improved using var2 and then it is used as the initial population of the GA.
- **var5:** To further improve the quality of the solution, FBI is used along with var4. This variant uses all steps of Algorithm 1.
- **COA:** For the purpose of comparison, modified datasets were also solved using a well-established algorithm, namely COA. The COA algorithm was developed for homogeneous activities, we modified the algorithm for singular activities. It is worth mentioning that the COA algorithm has previously outperformed many existing algorithms. The details of this algorithm and its parameters can be found in [32].

All of the above variants performance were tested using the same problem instances. Besides, for comparison, the same instances were also solved using a well-known existing algorithm, namely COA. For a fair comparison, we set the population size and maximum fitness evaluations to 10 and 5000, respectively, and all other parameters were kept the same as COA. All of the algorithms were run independently 31 times, to evaluate the performance and verify the robustness of the proposed algorithms. In each run, the evolution process is continue until it is reached the maximum fitness evaluations, i.e., MFFE = 5000. All algorithms were implemented in a Matlab2018b environment on a computer with a Core(TM)i7-8700 CPU@3.20GHz, and 16GB RAM, with Windows 10 operating system.

A. TEST PROBLEMS: MODIFIED PSPLIB BENCHMARKS

In this subsection, the proposed and existing algorithms were tested using the well-known PSPLIB [12] benchmarks with j30, j60, j90, and j120 instances, where j represents the number of non-dummy activities. As the original PSPLIB benchmarks were designed for homogeneous activities, where each activity may requires all of the available renewable resources, we modified the original instances as we assume that each activity requires only one renewable resource at a particular time instead of all of them, while keeping the resource availability same as they originally were. To generate new benchmark sets, the original instance has been modified as follows:

- **Case 1:** For each activity, keeping the highest resource demand as they are and the remaining demands converted to zero by using Eqn. (9). For example, if the resource demand of the j^{th} activity are $r_1 = 5$, $r_2 = 9$, $r_3 = 6$, and $r_4 = 2$, then the modified resource demand will be, $r_1 = 0$, $r_2 = 9$, $r_3 = 0$, and $r_4 = 0$.

$$r_{j,k} = \begin{cases} r_{j,k} = max(r_{j,K}), & \text{Keeping highest} \\ & \text{resource demand} \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

- **Case 2:** In some cases, an activity may require same highest amount of resources for two or more resource types. In that case, we take the first highest resource demand as it is and the remaining demands converted to zero. For example, if the resource demand of the j^{th} activity are $r_1 = 5$, $r_2 = 5$, $r_3 = 6$, and $r_4 = 6$, then the modified resource demand will be, $r_1 = 0$, $r_2 = 0$, $r_3 = 6$, and $r_4 = 0$.

To simulate real-world problems, the problem instances have been modified according to the singular resource requirements, and so optimal solutions of the new problem instances are unknown. Therefore, we evaluate the percentage of average deviation from the lower bound (LB) and the LB of each instance is calculated as the FT of the last dummy activity while only considering precedence constraints. The modified test instances were then used to evaluate the performance of each algorithm. After 31 independent runs, the performance is reported in Table 2-5 as a percentage deviation of mean makespan from the lower bound ($\%Dev_M$), average standard deviation ($avSTD$), the average deviation of minimum makespan from the lower bound ($\%Dev_{LB}$), and computational time in days. The $\%Dev_M$, $avSTD$, and $\%Dev_{LB}$ are calculated using Eqn. (10), Eqn. (11), and Eqn. (12), respectively.

$$\%Dev_M = 100 \times \frac{\sum_{ins=1}^{N} M_{Mspan,n} - \sum_{ins=1}^{N} LB_n}{\sum_{ins=1}^{N} LB_n} \quad (10)$$

$$avSTD = \frac{\sum_{ins=1}^{N} STD_n}{N} \quad (11)$$

$$\%Dev_{LB} = 100 \times \frac{\frac{\sum_{ins=1}^{N} Min_{Mspan,n} - \sum_{ins=1}^{N} LB_n}{\sum_{ins=1}^{N} LB_n}}{N} \quad (12)$$

where $M_{Mspan,n}$ and $Min_{Mspan,n}$ is the mean and minimum makespan of the n^{th} instance, respectively, LB_n is the critical path duration of the n^{th} instance (that does not consider resource constraint), and N is the total number of instances, i.e., 480, 480, 480, and 600 for j30, j60, j90, and j120, respectively.

TABLE 2. Comparison of the experimental results obtained from the proposed approach and COA for j30 instances.

Algorithm	$\%Dev_{Mean}$	avSTD	$\%Dev_{LB}$	Time (Days)
COA	4.490515	0.003458	4.795271	2.29
var1	13.063282	2.894460	5.674771	4.98E-04
var2	6.216350	0.715471	5.121282	0.0052
var3	4.632273	0.121479	4.795271	0.44
var4	4.513069	0.028660	4.795271	0.49
var5	4.485533	0.001417	4.795271	1.33

TABLE 3. Comparison of the experimental results obtained for the proposed approach and COA for j60 instances.

Algorithm	$\%Dev_{Mean}$	avSTD	$\%Dev_{LB}$	Time (Days)
COA	2.736021	0.039604	2.815104	6.58
var1	12.387871	3.395115	5.662487	6.03E-04
var2	4.686271	0.930156	3.198306	0.04
var3	3.205240	0.311868	2.844206	0.91
var4	2.930617	0.154375	2.829896	4.01
var5	2.731030	0.030167	2.815104	5.71

Table 2-5, summarizes the experimental results for the benchmarks sets of j30, j60, j90, and j120. It can be seen that deviation and average STD of var1 and var2 are higher than var3, var4, var5, and COA for all of the benchmark instances. However, the time required to find the solution of var1 and var2 is significantly lower than the other variants of the proposed approach and COA. Note that the way COA was implemented for solving the new test problems, it would not be computationally efficient for these problems. In other words, COA will always take longer time than any of the custom designed algorithm implemented in this research. So, the time comparison with COA is not really a fair comparison but it can be considered as a worst case time

TABLE 4. Comparison of the experimental results obtained from the proposed approach and COA for j90 instances.

Algorithm	$\%Dev_{Mean}$	avSTD	$\%Dev_{LB}$	Time (Days)
COA	1.707637	0.091156	1.666741	10.92
var1	11.539480	3.471952	5.494250	0.001
var2	3.900439	0.922968	2.426450	0.16
var3	2.523390	0.423523	1.922311	1.52
var4	2.157864	0.234264	1.827903	5.34
var5	1.667426	0.052764	1.662803	8.63

TABLE 5. Comparison of the experimental results obtained from the proposed approach and COA for j120 instances.

Algorithm	$\%Dev_{Mean}$	avSTD	$\%Dev_{LB}$	Time (Days)
COA	7.800298	0.329061	7.557719	26.56
var1	30.796911	7.323794	18.504604	0.0019
var2	13.533665	2.145715	10.264670	0.57
var3	10.374164	1.069706	8.813073	3.19
var4	9.533994	0.692954	8.503703	12.99
var5	7.753976	0.267065	7.491923	20.38

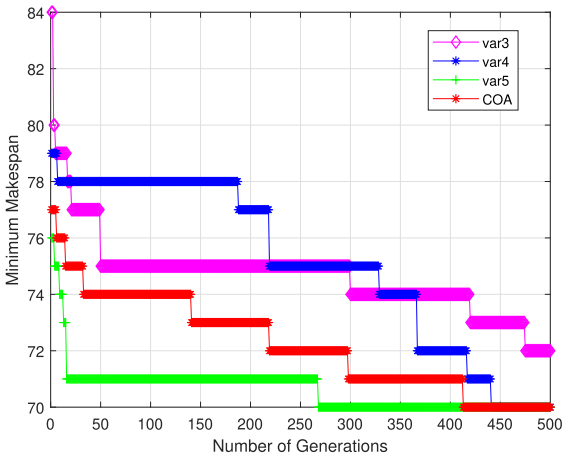


FIGURE 5. Convergence plot of minimum makespan for an instance.

scenario. However, it helps to validate the quality of solutions. The reason for considering COA is that we do not have any other algorithm handy for result validation and comparison. The performance of var5 and COA are almost similar for small-scale test problems, i.e., j30, and j60 benchmarks, however, var5 outperformed COA for large-scale test problem, i.e., j90, and j120. Moreover, the computational budget of var5 is also significantly lower than COA.

The proposed approach demonstrates that it can obtain best quality solutions with less computational time than that of the existing state-of-the-art algorithm. Fig. 5 shows the convergence rate of var3, var4, var5, and COA. It is seen than var5 converges faster than the other variants of the proposed approach and COA. It is also seen that the proposed approach performs better even for the wide-range benchmarks and the overall ranking of the proposed one is the best one that is discussed in subsection V-D1.

B. COMPARISON WITH EXACT SOLVER

In this subsection, to judge the quality of solutions obtained from our algorithm, we solved some randomly selected instances from the test sets using the Branch and Cut (B&C) algorithm built in Matlab as OPTI toolbox. For the experiment, 10 instances from each of 30, 60, 90, and 120 activities problems were randomly selected and they were solved using our proposed approach as well as the exact solver (B&C). To solve the selected problems using B&C algorithm, we need less than 500 nodes for j30 problems, 81k for j60, and 213k for j90. However, we were unable to obtain any feasible solution for j120 problems even with up to 2 million nodes. For j30, j60 and j90 instances, there are no differences between the objective function values of these two methods. The computational times of the two approaches are reported in Table 6. The maximum number of nodes required to find the solution by the B&C algorithm are also reported in the last column of the table. From Table 6 and Fig. 6, it is found that time required by the exact method is lower than the proposed method for 30 activities problem, however, it is increased almost exponentially for 60, and 90 activities problems.

TABLE 6. Comparison of computational times of the proposed approach and exact method.

Benchmark	Average Computational Time		Maximum Nodes
	Proposed	Exact	
j30	14.57	2.03	472
j60	33.13	206.99	80672
j90	51.11	545.74	212771
j120	90.76	-	2 million

TABLE 7. Effect of different population size.

PS	Time (seconds)							
	j30		j60		j90		j120	
	Proposed	COA	Proposed	COA	Proposed	COA	Proposed	COA
5	14.87	16.07	33.63	36.04	52.08	56.55	93.41	97.41
10	**14.57**	15.27	**33.13**	35.73	**51.11**	54.32	90.76	94.67
15	15.31	15.56	33.47	35.87	51.46	54.68	90.92	94.60
20	15.51	16.10	33.79	37.08	51.46	54.72	91.70	93.71
25	15.54	16.87	34.16	36.19	51.89	54.78	**90.67**	94.03

The computational budget of the proposed approach is increased linearly for the small to large scale problem, which demonstrates the consistency of the proposed approach.

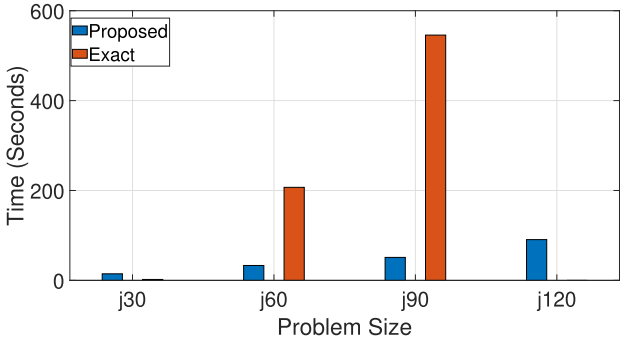


FIGURE 6. Computational time comparison for proposed and exact method.

C. PARAMETRIC TEST

In this subsection, the effect of different parameters: (1) effect of problem size, (2) effect of population size, (3) effect of maximum fitness evaluation, and (4) effect of considering local search to generate the feasible solution, are analyzed.

1) EFFECT OF PROBLEM SIZE

In this experiment, the effect of problem size is shown in Fig. 7. From that figure, it is found that the computational budget is increased while increasing the problem size (time is shown in Table 2-5 in days), which is highly nonlinear. However, the proposed approach takes considerably less time than that of COA, which demonstrate the effectiveness of the proposed approach and ensures that the proposed approach converges quickly towards the best quality solution.

2) EFFECT OF POPULATION SIZE

In this experiment, we chose five different population sizes as: n, 2n, 3n, 4n, and 5n (where n = 5), to analyze the effect of various population sizes. The experiment was carried out on some selected instances from each of the benchmark problems and the effect of fitness value and computational time are shown in Fig. 8 and Table 7 respectively. It is found that increasing the population size may result in better mean makespan (Fig. 8), however computational time is also increased (Table 7). From the experiment, it is found that the population size of 2n is the best among the five options. In comparison with COA, the proposed approach showed less variance of makespan than that of COA. Furthermore, the proposed approach can find the same quality of solution with less computational budget which ensures the quicker convergence rate of the proposed approach than that of COA.

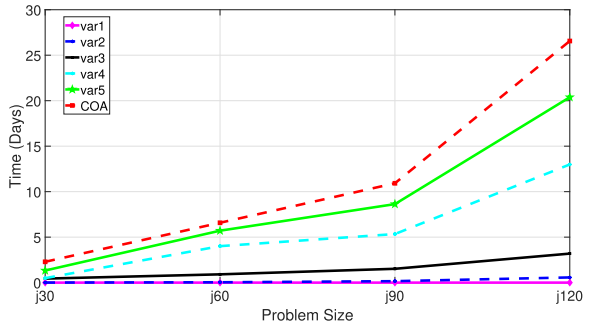


FIGURE 7. Effect of problem size.

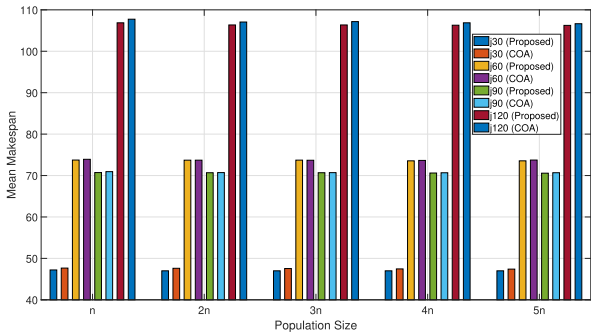


FIGURE 8. Effect of population size.

3) EFFECT OF MAXIMUM FITNESS EVALUATIONS

In this subsection, the maximum fitness evaluation is varied and the effect of this experiment is reported in Table 8. In our experiment, we mainly considered 10 as population size and 5000 as the number of maximum fitness evaluations. For experimental analysis, we considered the performance when evaluations reached 1000 and this analysis was evaluated using all benchmarks instances.

From Table 8, it is clear that the performance of the proposed algorithm is improved when the experiment is allowed a higher number of fitness evaluations. Algorithms with 5000 fitness evaluations outperformed for all benchmarks over with 1000 fitness evaluations. It is worth mentioning that the proposed approach outperformed COA regardless of the number of fitness evaluations.

TABLE 8. %avDEV when maximum fitness evaluation varies.

Benchmark	avDEV							
	var3		var4		var5		COA	
	1k	5k	1k	5k	1k	5k	1k	5k
j30	4.796174	4.795271	4.795376	4.795271	4.795376	4.795271	4.795376	4.795271
j60	2.945818	2.844206	2.846750	2.829896	2.823868	2.815104	2.835286	2.815104
j90	2.189889	1.922311	1.940791	1.827903	1.664615	1.662803	1.680831	1.666741
j120	9.694013	8.813073	8.963198	8.503703	7.530534	7.491923	7.624154	7.557719

TABLE 9. Effect of local search.

Benchmark	Instances	Average Deviation from Lower Bound					
		Without		With NS		With FBI	
		Min	Mean	Min	Mean	Min	Mean
j30	7, 124, 208, 322, and 451	6.69145	7.315985	6.69145	6.992565	6.69145	6.69145
j60	2, 121, 208, 325, and 450	10.875332	13.700265	10.875332	11.763926	10.875332	10.883289
j90	4, 164, 206, 369, and 401	6.888889	10.717778	6.888889	8.497778	6.444444	7.246667
i120	58, 155, 208, 307, and 451	18.899083	26.06055	17.247706	20.933945	13.944954	15.963303

4) EFFECT OF CONSIDERING LOCAL SEARCH

In this experiment, we examined two local search mechanisms i.e., (1) neighbourhood swapping, and (2) forward-backward improvement along with the proposed algorithm. For the experimental study, five different complex instances are taken from each of the benchmarks as stated in Table 9.

The average deviation of minimum and mean makespan from the lower bound, with or without local search, is shown in Table 9. It is found that the deviation of minimum and mean makespan are similar for the small-scale benchmark problem. However, the average deviation of the proposed approach increases significantly for the large-scale problems where local search is not considered. This mainly happens due to the algorithm without local searches becoming stuck in local optimum for the large-scale benchmarks.

D. STATISTICAL TEST

In this subsection, the proposed approach has been validated by using different statistical tests. To do this we use (1) Friedman test ranking, (2) Wilcoxon test, and (3) Boxplot, to compare the results among the different proposed variants and COA.

1) FRIEDMAN TEST

A Friedman test is used to test the overall rank of an algorithm. In this experimental step, the average mean makespan deviation from the lower bound is used as input and the corresponding ranking is reported in Table 10. From the result, it is seen that var5 gets the highest ranking among all of the proposed variants and COA, which further indicates the superiority of the proposed algorithm.

2) WILCOXON TEST

A Wilcoxon test is performed to evaluate the performance of all the proposed variants and the objective values obtained from 31 runs were considered as sample data. The value of p indicates the significance of the samples, where $p < 0.05$ represents that there is a statistically significant difference between the samples. The result obtained from the Wilcoxon test is shown in Table 11, which shows that p values are less than the 0.05; this shows a statistically significant difference between the proposed approach and COA.

TABLE 10. Friedman test ranks for different algorithms.

Criteria	var1	var2	var3	var4	var5	COA
Mean Makespan	1.00	2.00	3.00	4.00	**6.00**	5.00

TABLE 11. Wilcoxon sign test results for different algorithms.

Algorithms	Criterion	Better	Similar	Worse	p-value
var5 vs var1	Mean Makespan	31	0	0	0.00
var5 vs var2	Mean Makespan	28	3	0	0.00
var5 vs var3	Mean Makespan	18	13	0	0.00
var5 vs var4	Mean Makespan	12	19	0	0.00
var5 vs COA	Mean Makespan	7	24	0	0.00

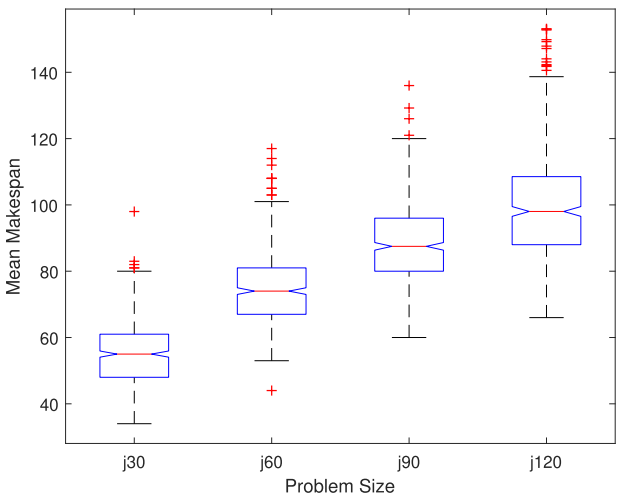


FIGURE 9. Boxplot of mean makespan of all benchmarks.

3) BOX PLOT

In this subsection, a boxplot is used to present the performance of the proposed algorithm. The mean fitness value of each individual instance of the benchmarks are used as the input data and the corresponding statistical result is shown in Fig. 9. The middle line of the box presents the median value of the fitness value for the corresponding benchmarks. The bottom and top lines represent the 25% and 75% percentile of the final values, respectively. From Fig. 9, it is seen that most of the mean makespan of each instance is close to the mean result for the corresponding benchmarks, which shows that S-GA obtained a good quality result for each instance of each benchmark.

VI. SUMMARY AND CONCLUSION

Despite the large number of solution approaches to solve RCPSPs over the past few decades, they are designed for homogeneous activities where each activity may require several of the available resource types. However, real-world problems do not require every resource all the time, usually need one specific resource for one specific activity. Therefore, we considered RCPSPs with singular activities and modified the standard benchmarks where each activity needs only one resource type, instead of all. To solve this variant of RCPSPs, we proposed an S-GA, that is based on a GA and three heuristics.

The performance of the proposed framework is evaluated by solving modified test problems with up to 120 non-dummy activities. As we modified the original benchmarks, the same problems were solved with the well-known COA algorithm. Furthermore, different statistical tests and parametric analysis were performed to investigate the effectiveness of the proposed variants of S-GA. From the experimental result, the findings can be summarized as:

- S-GA outperformed COA; S-GA can find better quality solutions.
- From the statistical tests, it is seen that our proposed approach is the best in comparison to COA.
- The inclusion of local search enhances the performance of S-GA.
- The proposed approach is more consistent than that of COA, regarding the effect of maximum fitness evaluations.
- The effect of population size has only a small impact on the performance.
- The exact solver ensures that S-GA obtains the same solution for the small to large scale problem.

In future, this research can be extended by considering uncertainty in activity duration. In addition, time-varying resource availability and resource requirements would be another potential area to explore.

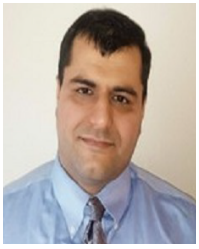
REFERENCES

[1] H. F. Rahman, R. K. Chakrabortty, and M. J. Ryan, "Memetic algorithm for solving resource constrained project scheduling problems," *Autom. Construct.*, vol. 111, Mar. 2020, Art. no. 103052.
[2] P. Brucker, "Scheduling and constraint propagation," *Discrete Appl. Math.*, vol. 123, nos. 1–3, pp. 227–256, Nov. 2002.
[3] J. Blazewicz, J. K. Lenstra, and A. H. G. R. Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Appl. Math.*, vol. 5, no. 1, pp. 11–24, Jan. 1983.
[4] F. Mahmud, F. Zaman, R. Sarker, and D. Essam, "Heuristic embedded genetic algorithm for heterogeneous project scheduling problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2020, pp. 1–8.
[5] F. Zaman, S. Elsayed, R. Sarker, D. Essam, and C. A. C. Coello, "An evolutionary approach for resource constrained project scheduling with uncertain changes," *Comput. Oper. Res.*, vol. 125, Jan. 2021, Art. no. 105104.
[6] S. M. Elsayed, R. A. Sarker, and D. L. Essam, "A new genetic algorithm for solving optimization problems," *Eng. Appl. Artif. Intell.*, vol. 27, pp. 57–69, Jan. 2014.
[7] F. Zaman, S. M. Elsayed, R. Saker, and D. Essam, "Resource constrained project scheduling with dynamic disruption recovery," *IEEE Access*, vol. 8, pp. 144866–144879, 2020.
[8] V. Valls, F. Ballestín, and S. Quintanilla, "A hybrid genetic algorithm for the resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 185, no. 2, pp. 495–508, Mar. 2008.
[9] D. Debels and M. Vanhoucke, "A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem," *Oper. Res.*, vol. 55, no. 3, pp. 457–469, 2007.
[10] S. M. Elsayed, R. A. Sarker, and D. L. Essam, "A three-strategy based differential evolution algorithm for constrained optimization," in *Proc. Int. Conf. Neural Inf. Process.* Berlin, Germany: Springer, 2010, pp. 585–592.
[11] R. K. Chakrabortty, A. Abbasi, and M. J. Ryan, "Multi-mode resource-constrained project scheduling using modified variable neighborhood search heuristic," *Int. Trans. Oper. Res.*, vol. 27, no. 1, pp. 138–167, Jan. 2020.
[12] R. Kolisch and A. Sprecher, "PSPLIB—A project scheduling problem library: OR software-ORSEP operations research software exchange program," *Eur. J. Oper. Res.*, vol. 96, no. 1, pp. 205–216, 1997.
[13] D. Recalde, R. Torres, and P. Vaca, "An exact approach for the multi-constraint graph partitioning problem," *EURO J. Comput. Optim.*, vol. 8, nos. 3–4, pp. 289–308, Oct. 2020.
[14] M. Ritt and A. M. Costa, "Improved integer programming models for simple assembly line balancing and related problems," *Int. Trans. Oper. Res.*, vol. 25, no. 4, pp. 1345–1359, Jul. 2018.
[15] S. Al-Shihabi and M. M. AlDurgam, "The contractor time–cost–credit trade-off problem: Integer programming model, heuristic solution, and business insights," *Int. Trans. Oper. Res.*, vol. 27, no. 6, pp. 2841–2877, Nov. 2020.
[16] C. Reintjes and U. Lorenz, "Bridging mixed integer linear programming for truss topology optimization and additive manufacturing," *Optim. Eng.*, vol. 22, no. 2, pp. 1–45, 2020.
[17] P. Brucker, S. Knust, A. Schoo, and O. Thiele, "A branch and bound algorithm for the resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 107, no. 2, pp. 272–288, Jun. 1998.
[18] Z. Pei, M. Wan, Z.-Z. Jiang, Z. Wang, and X. Dai, "An approximation algorithm for unrelated parallel machine scheduling under TOU electricity tariffs," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 2, pp. 743–756, Apr. 2021.
[19] N. Bianchessi, R. Mansini, and M. G. Speranza, "A branch-and-cut algorithm for the team orienteering problem," *Int. Trans. Oper. Res.*, vol. 25, no. 2, pp. 627–635, Mar. 2018.
[20] M. Colvin and C. T. Maravelias, "Modeling methods and a branch and cut algorithm for pharmaceutical clinical trial planning using stochastic programming," *Eur. J. Oper. Res.*, vol. 203, no. 1, pp. 205–215, May 2010.
[21] F. Zaman, S. Elsayed, R. Sarker, D. Essam, and C. A. C. Coello, "Evolutionary algorithm for project scheduling under irregular resource changes," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 403–410.
[22] P. I. Adamu, H. I. Okagbue, and P. E. Oguntunde, "A new priority rule for solving project scheduling problems," *Wireless Pers. Commun.*, vol. 106, no. 2, pp. 681–699, May 2019.
[23] R. Klein, "Bidirectional planning: Improving priority rule-based heuristics for scheduling resource-constrained projects," *Eur. J. Oper. Res.*, vol. 127, no. 3, pp. 619–638, Dec. 2000.
[24] R. Zamani, "A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 229, no. 2, pp. 552–559, Sep. 2013.

[25] M. Asadujjaman, H. F. Rahman, R. K. Chakrabortty, and M. J. Ryan, "An immune genetic algorithm for solving NPV-based resource constrained project scheduling problem," *IEEE Access*, vol. 9, pp. 26177–26195, 2021.
[26] J. Alcaraz and C. Maroto, "A robust genetic algorithm for resource allocation in project scheduling," *Ann. Oper. Res.*, vol. 102, nos. 1–4, pp. 83–109, 2001.
[27] R. Zamani, "An evolutionary implicit enumeration procedure for solving the resource-constrained project scheduling problem," *Int. Trans. Oper. Res.*, vol. 24, no. 6, pp. 1525–1547, Nov. 2017.
[28] F. Kong and D. Dou, "RCPSP with combined precedence relations and resource calendars," *J. Construct. Eng. Manage.*, vol. 146, no. 12, Dec. 2020, Art. no. 04020133.
[29] M.-Y. Cheng, D.-H. Tran, and Y.-W. Wu, "Using a fuzzy clustering chaotic-based differential evolution with serial method to solve resource-constrained project scheduling problems," *Autom. Construct.*, vol. 37, no. 1, pp. 88–97, 2011.
[30] H. Zhang, X. Li, H. Li, and F. Huang, "Particle swarm optimization-based schemes for resource-constrained project scheduling," *Autom. Construct.*, vol. 14, no. 3, pp. 393–404, Jun. 2005.
[31] Ö. H. Bettemir and R. Sonmez, "Hybrid genetic algorithm with simulated annealing for resource-constrained project scheduling," *J. Manage. Eng.*, vol. 31, no. 5, Sep. 2015, Art. no. 04014082.
[32] S. Elsayed, R. Sarker, T. Ray, and C. C. Coello, "Consolidated optimization algorithm for resource-constrained project scheduling problems," *Inf. Sci.*, vols. 418–419, pp. 346–362, Dec. 2017.
[33] F. Zaman, S. Elsayed, R. Sarker, and D. Essam, "Scenario-based solution approach for uncertain resource constrained scheduling problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.
[34] C. Fang and L. Wang, "An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem," *Comput. Oper. Res.*, vol. 39, no. 5, pp. 890–901, May 2012.
[35] K. Ziarati, R. Akbari, and V. Zeighami, "On the performance of bee algorithms for resource-constrained project scheduling problem," *Appl. Soft Comput.*, vol. 11, no. 4, pp. 3720–3733, Jun. 2011.
[36] F. Mahmud, F. Zaman, R. Sarker, and D. Essam, "Memetic algorithm for heterogeneous project scheduling problems," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2020, pp. 1–8.
[37] N. Christofides, R. Alvarez-Valdés, and J. M. Tamarit, "Project scheduling with resource constraints: A branch and bound approach," *Eur. J. Oper. Res.*, vol. 29, no. 3, pp. 262–273, Jun. 1987.
[38] J. E. Kelley, "The critical-path method: Resource planning and scheduling," *Ind. Scheduling*, vol. 13, no. 1, pp. 347–365, 1963.
[39] S. Rostami, S. Creemers, and R. Leus, "New strategies for stochastic resource-constrained project scheduling," *J. Scheduling*, vol. 21, no. 3, pp. 349–365, Jun. 2018.
[40] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, no. 2, pp. 95–99, Aug. 1988.
[41] V. Valls, F. Ballestın, and S. Quintanilla, "Justification and RCPSP: A technique that pays," *Eur. J. Oper. Res.*, vol. 165, no. 2, pp. 375–386, Sep. 2005.
[42] K. Y. Li and R. J. Willis, "An iterative scheduling technique for resource-constrained project scheduling," *Eur. J. Oper. Res.*, vol. 56, no. 3, pp. 370–379, Feb. 1992.

FIROZ MAHMUD received the B.Sc. and M.Sc. degrees in computer science and engineering from Rajshahi University of Engineering and Technology (RUET), Rajshahi, Bangladesh, in 2009 and 2017, respectively. He is currently pursuing the Ph.D. degree with the School of Engineering and Information Technology, University of New South Wales (UNSW) Canberra, Canberra, ACT, Australia. His current research interests include constrained optimization, evolutionary algorithms, multiobjective optimization, and machine learning.

FORHAD ZAMAN received the Ph.D. degree from the University of New South Wales (UNSW) Canberra, Canberra, ACT, Australia, in 2017. He worked as a Research Fellow with UNSW Canberra, from 2017 to 2019, where he is currently an Adjunct Lecturer with the School of Engineering and Information Technology. His current research interests include power system operation and control, active bidding strategy in the electricity market, evolutionary algorithms, multiobjective optimization, handling real-life uncertainty, and game theory.



ALI AHRARI received the Ph.D. degree in mechanical engineering from Michigan State University, East Lansing, MI, USA, in 2016. He is currently a Postdoctoral Research Associate with the University of New South Wales Canberra, Canberra, ACT, Australia. His research interests include evolutionary algorithms and engineering optimization. He is currently a member of the IEEE CIS Task Force on Multimodal Optimization. He has won GECCO Competition in multimodal optimization, in 2016 and 2017, and the ISCSO Student Competition in structural optimization, in 2017 and 2018.

RUHUL SARKER (Member, IEEE) received the Ph.D. degree from Dalhousie University, Halifax, NS, Canada, in 1992. He is currently a Professor with the School of Engineering and IT, and the former Director with the Faculty Postgraduate Research, University of New South Wales Canberra, Canberra, ACT, Australia. He has authored a book titled *Optimization Modelling: A Practical Approach*. His current research interests include evolutionary optimization and applied operations research. He is a member of INFORMS.

DARYL ESSAM (Member, IEEE) received the B.Sc. degree in computer science from the University of New England, Australia, in 1990, and the Ph.D. degree from the University of New South Wales, Australia, in 2000. Since 1994, he has been with UNSW Canberra, where he is currently a Senior Lecturer. His research interests include genetic algorithms, with a focus on both evolutionary optimization and large scale problems.