A Heuristic Algorithm for Solving Resource Constrained Project Scheduling Problems

Shelvin Chand[1], Hemant Kumar Singh[2], Tapabrata Ray[3]
School of Engineering & Information Technology
University of New South Wales
Canberra Australia
Email: Shelvin.Chand@student.adfa.edu.au[1], H.Singh@adfa.edu.au[2], T.Ray@adfa.edu.au[3]

***Abstract*—Resource constrained project scheduling problem (RCPSP) is one of the classical problems in the area of discrete optimization. In this paper we propose an algorithm for solving RCPSP which relies on an adaptive insertion mutation operator that targets different regions of the search space and attempts to exploit neighborhoods via forward-backward iterative local search. Furthermore, the algorithm makes use of an archive to ensure better utilization of the schedule budget. The performance of the approach is analyzed across various problem complexities associated with J30, J60 and J120 full instance sets of PSPLib with budgets of 1,000, 5,000 and 50,000 schedules. The study provides insights on the performance of the algorithm i.e. *why the performance is good for particular instances and not as good for others*.**

I. Introduction

The classical resource constrained project scheduling problem (RCPSP) is a combinatorial optimization problem which belongs to the class of NP-hard problems. It extends the commonly encountered task of project management to include resource constraints which limit the number of tasks which can be executed at any given time. Research on RCPSP is particularly important since such resource constrained problems are encountered in almost every manufacturing environment.

Exact approaches such as linear programming [21] are typically only used for smaller problem instances since the computational time increases exponentially with the size of the problem. This has prompted the development of heuristic and meta-heuristic approaches. Heuristic approaches for RCPSP include ones based on genetic algorithms [31], artificial bee colony [15], particle swarm optimization [6], artificial immune systems [2], ant colony optimization [24], scatter search [9], tabu search [25], multi-agent optimization [32] etc. In addition to such approaches there also exist a number of priority rules/functions [17] which can be used as a rule-of-thumb in the absence of sophisticated algorithms. Priority rules are particularly useful in cases where cheaper, more intuitive solutions are preferred over optimality, something that is commonly seen in smaller manufacturing environments. There exist countless algorithms within literature for RCPSP therefore it is almost impossible to cover all within the scope of this paper due to space limitations. However, the interested reader can refer to [14] and [19] for a detailed analysis on the state-of-the-art.

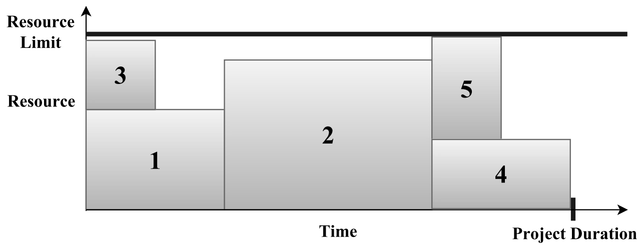


Fig. 1: Visualization of generated schedule.

In this paper, we propose a simple population based stochastic algorithm for solving the RCPSP. The algorithm uses an adaptive insertion operator which explores new regions by traversing through infeasible regions of the search space. A new region of the search space is then exploited using the well established forward-backward iterative local search. Archives are maintained to ensure only unique new solutions (activity lists) are evaluated. Furthermore, to provide greater insights on the performance of the algorithm, a detailed analysis is undertaken across problem instances of J30, J60 and J120. This study is particularly valuable as it establishes the link between problem complexity and the performance of the algorithm.

The rest of the paper is organized as follows. Section II gives a background discussion on the problem. Section III describes the proposed algorithm while section IV gives details on the effect of the algorithm parameters. A detailed discussion on the links between problem complexity and the performance of the algorithm is presented in section V. Finally, section VI summarizes the findings and lists several potential research directions.

II. Problem Definition

The resource constrained project scheduling problem is classified as NP-hard. Each project consists of a set of activities that need to be scheduled while considering precedence and resource constraints. If activity j is a successor of activity i then activity i must be completed before activity j can be started. All such relationships among the order of activities constitute the precedence constraints.

Each project is also assigned a set of renewable resources which are available in given capacities for the entire duration of the project. Each activity may require one of more of these resources to be completed. While scheduling the activities, the daily resource usage limits can not be exceeded, which is referred to as the resource constraint. The overall goal of the problem is to minimize the *makespan*, which refers to the total duration from the start of first to the end of last scheduled activity. The visualization of a schedule is given in Fig. 1.

III. Algorithm Details

Following from the above discussion, we attempt to design an algorithm which offers competitive and robust performance across problems of different sizes and complexity. The following sub-sections present the details of the proposed algorithm (Algorithm 1).

A. Solution Representation

In this work we have adopted the activity list representation. Each solution is a 1 x *m (m=no. of activities)* vector containing all the activity ids (including dummy activities). The order of the activities represents the priority of each activity. A given solution is decoded using the serial Schedule Generation Scheme (SGS) [18]. The serial SGS is used for both the forward and backward schedules generated by the algorithm. Generating a schedule involves assigning start and finish times to activities based on the resource availability and precedence constraints.

Algorithm 1 Proposed Algorithm for Solving RCPSP

```
1: Define: $n$: PopSize, $ip$: Insertion%, $sch_{max}$: Total Schedules
2: $pop$: Population = GenerateInitialPop($n$) {Refer Algorithm 2}
3: while $sch_{eval} <= sch_{max}$ do
4:     $p$: Parents = ThreeWayTournamentSelection($pop$)
5:     for $i$ = 1:$n$ do
6:         $sol_{mut}$ = InsertionMutation($p$, $ip$) {Refer Algorithm 3}
7:         $sol_{ls}$ = ForwardBackwardIteration($sol_{mut}$)
8:         AddToPopulation($sol_{ls}$)
9:     end for
10:    $pop_{sorted}$ = Sort($pop$)
11:    $pop$ = Trim($pop_{sorted}$)
12: end while
```

Algorithm 2 Initialization

```
1: Define: $n$:PopSize
2: for $i$ = 1 to $n$ do
3:     Create 1x$m$ Vector Containing Activity IDs
4:     Randomly Shuffle the Solution Vector
5:     Repair Solution
6:     Generate Forward Schedule using Repaired Solution
7:     Add solution to the Population
8: end for
```

B. Initial Population

The initialization (Algorithm 2) starts by generating a vector of size *m (m=no. of activities)* containing all the activity ids which are ordered from lowest to highest. This vector is then randomly shuffled without taking into account any form of precedence relationship resulting in a randomly initialized population. Every solution is repaired using Algorithm 4 to conform to the precedence constraints. The precedence feasible repaired solution is then evaluated using forward scheduling resulting in a feasible schedule that satisfies all constraints of the RCPSP problem.

Algorithm 3 Adaptive Insertion Mutation

```
1: Define: $ip$:Insertion Percentage, $sch_{eval}$:Schedules Evaluated, $sch_{max}$Total
   Schedules, $m$: Solution Size
2: $numIns$:Number of Insertions = $m * ip$;
3: for i = 1 to $numIns$ do
4:     Randomly Select an Activity: $l_1$: Location1
5:     $Diff_L$: Difference Left = $l_1$; $Diff_R$: Difference Right = $m$-$l_1$;
6:     $Move_{per}$: Percentage Movement = (1-($sch_{eval}$/$sch_{max}$))
7:     $w$: Window = $Move_{per}$*$m$
8:     Randomly generate: $Move_{prob}$: Movement Probability
9:     if $Move_{prob}$ >=0.5 OR $Diff_L$ = 0 then
10:        if $w > Diff_R$ then
11:            $w$ = $Diff_R$
12:        end if
13:        if $w$ = 0 OR $w$ = 1 then
14:            $l_2$: Location2 = $l_1$+1
15:        else
16:            $ub$: Upper Bound=$l_1$+$w$; $lb$:Lower Bound = $l_1$+1;
17:            $l_2$ = Random Number between $ub$ and $lb$ (inclusive)
18:        end if
19:    else if $Move_{prob}$ <0.5 OR $Diff_R$ = 0 then
20:        if $w > Diff_L$ then
21:            $w$ = $Diff_L$
22:        end if
23:        if $w$ = 0 OR $w$ = 1 then
24:            $l_2$ = $l_1$-1
25:        else
26:            $ub$ = $l_1$-1; $lb$ = $l_1$-$w$;
27:            $l_2$ = Random Number between $ub$ and $lb$ (inclusive)
28:        end if
29:    end if
30:    Move Activity at $l_1$ to $l_2$
31: end for
32: Repair Solution {Refer Algorithm 4}
```

Algorithm 4 Repair Mechanism

```
1: Define: $sol_{inf}$: Infeasible Solution, $sol_f = \emptyset$: Repaired Feasible Solution, $m =$
   $size(sol_{inf})$: Solution Size
2: for $i$ = 1:$m$ do
3:     for $j$ = 1:size($sol_{inf}$) do
4:         if All predecessors of $sol_{inf}[j]$ are in $sol_f$ then
5:             Insert $sol_{inf}[j]$ into $sol_f$ at location $i$
6:             Remove activity at location $j$ from $sol_{inf}$
7:             $break$ inner loop
8:         end if
9:     end for
10: end for
```

C. Selection

Parent selection is an important aspect that affects the performance of all population based stochastic algorithms. Parent selection and recombination operators work collectively maintaining the fine balance between convergence and diversity. In the proposed algorithm, a three way tournament is used to identify parents that would participate in the generation of offspring via insertion mutation. A three-way tournament is more greedy as compared to a binary tournament and often necessary to induce an appropriate selection pressure.

This works similar to the binary tournament selection but with an additional solution considered in the tournament. Among the three solutions the fittest is selected for mutation.

D. Adaptive Insertion Operator

The proposed algorithm makes use of an insertion operator (Algorithm 3) based on *window of movement* to determine the new location of an activity. This was originally proposed in [5]. A random activity is selected from the activity list, and the available locations to the right and left of the activity is identified. In earlier phases of the search, the chosen activity can move across the entire span (left or right), while progressively this movement is reduced. The rationale behind this is that during the earlier generations one would want the algorithm to be able to explore different regions of the search space but in the later generations one would ideally want it to just explore the neighborhood of the current solutions.

The maximum length of movement is expressed as $(1 - (sch_{eval}/sch_{max}))$. This is multiplied by the length of the activity list to determine the actual number of slots (window) a particular activity can jump on either side of its location. Based on a random probability, the movement will either be on the right or left of the activity's location. If the window size is greater than the number of slots available in the direction of movement, then the window size is trimmed to be the same as the number of available slots. It is important to take note that after a jump, the solution might not satisfy the precedence constraint and thus would require a repair procedure to satisfy the feasibility. These infeasible moves aid the algorithm to avoid getting stuck in local minima. The user defined parameter $Insertion\%$ determines the number of insertion operations that will be performed for each activity list.

E. Local Search

All solutions generated using the mutation operation undergo local search. The adaptive insertion mutation generates solutions within promising regions of the search space and the local search explores these regions in greater depth. The local search procedure used in this study is based on the scheme proposed by Li and Willis [22].

The local search procedure, referred in literature as forward-backward iteration (FBI), performs an iterative forward and backward scheduling process until no improvement is possible. It begins by evaluating a solution using forward scheduling. The activities are then sorted in ascending order by their finish times (on the forward schedule) and a new activity list is generated. This new activity list is evaluated using backward scheduling. Following this, forward scheduling is applied on a new activity list which is generated by sorting the activities in ascending order based on their start times in the backward schedule. This iterative procedure will only result in solutions which have equal or better makespan in comparison to the current solution. Each schedule generated using FBI is counted in the function evaluations.

F. Replacement

After the mutation and local search, the population now contains more than 2*n* solutions. The population is sorted according to the makespan and the best *n* solutions are carried forward to the next generation.

G. Repair Mechanism

A solution repair mechanism (Algorithm 4) is used to repair any precedence infeasible solutions encountered during search. To repair an infeasible solution (activity list) with m activities, the procedure undertakes m passes over the activity list and constructs a repaired solution by removing elements one by one from the infeasible solution. In each pass, it will remove the first activity that has all its predecessors scheduled (already inserted into repaired solution) and add it to the repaired activity list vector. This will continue until the infeasible solution vector is completely empty and the repaired solution vector is filled with m activities.

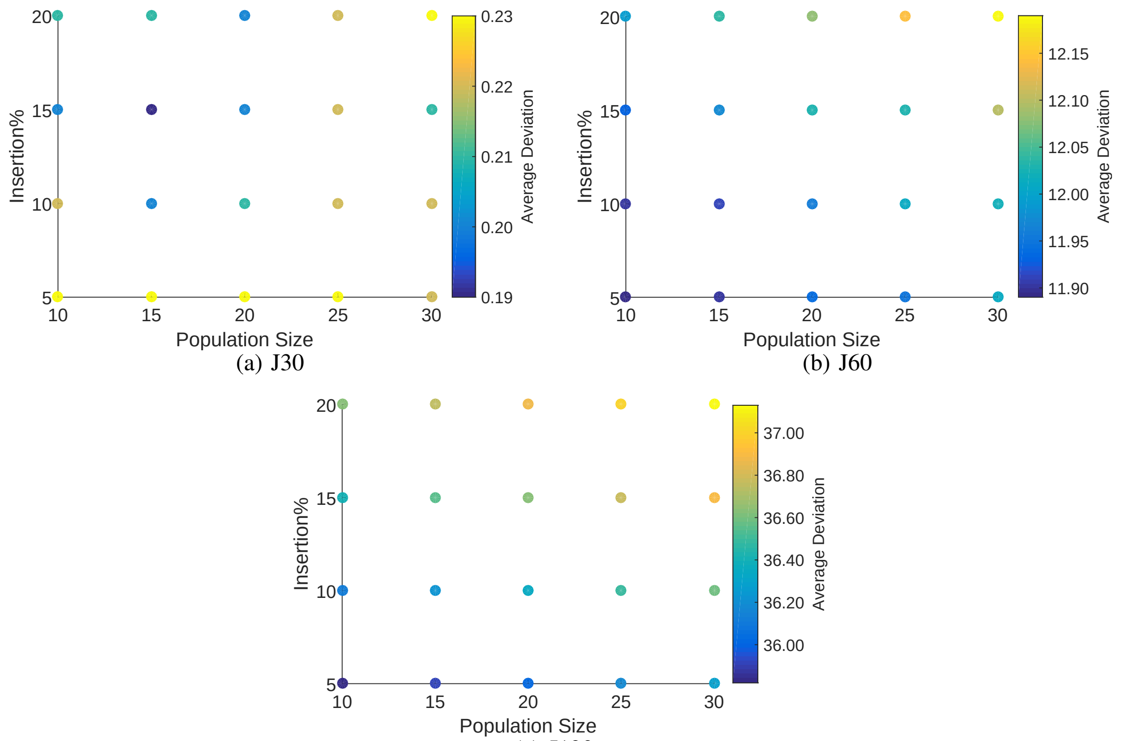
H. Activity List and Start Time Archives

Two separate archives are maintained for storing all the activity lists and start-times encountered through the course of evolution. The activity list archive stores all the solutions evaluated so far. Each solution before being evaluated is checked against the archive. If the solution exists within the archive, it will not be evaluated and the local search will terminate at that point. Since the evaluation budget is limited, this mechanism ensures that each evaluation is on a new unique solution.

A number of solutions are generated through local search. All the solutions are added to the population. The start-times for each evaluated solution are stored in an archive. If a generated schedule has an exact match with any vector in the start-time archive the corresponding activity list, the solution is not added into the population. This ensures two things. Firstly, the population will not have any repeated solutions both in-terms of activity lists and schedules. Secondly, this prevents the algorithm from wasting computational effort on schedules which have already been explored.

IV. Parameter Settings

A number of experiments were done to determine a suitable population size and insertion percentage. Trial runs were done with values from 10 to 30 (with intervals of 5) for population size and 5 to 20% (with intervals of 5%) for insertion percentage. This resulted in 20 different combinations. The algorithm was run across all instances of J30, J60 and J120 for 1,000 schedules. The obtained results are given in Figure 2. The results indicate that population size of 10 and insertion percentage of 5% give the best results (median) across majority of the problems. Results for J30 fluctuate across different parameters. Lower end of the parameter(both) values did not seem to give favourable results. For J60 the combinations on the lower left corner(subfigure (b)) give the best performance. These combinations give pretty much the same performance with slight but insignificant difference($\leq$ 0.01). For J120 population size of 10 and insertion percentage of 5% gave the best results. Hence we chose this combination as it gives good performance on J60 as well and reasonable performance on J30. A smaller population size offers the algorithm to evolve over more generations which appears to be more effective. A lower value of insertion percentage seems to work better i.e. maintains a balance between disruptive mutation and adequate exploration.



(a) J30 (b) J60

(c) J120

Fig. 2: Parameter effect on average percentage deviation

V. Computational Experiments

This section provides the details of the numerical experiments. The algorithm was developed using the Java programming language and all experiments were carried out on a Windows 7 computer with Intel i7 3.4GHz CPU and 16gb RAM. The standard PSPLib test problems i.e. J30, J60 and J120 [20] were used. J30 and J60 consist of 480 instances with 4 renewable resources and 30 and 60 non-dummy activities, respectively. J120 consists of 600 instances with 4 renewable resources and 120 non-dummy activities. For each problem, the algorithm was run with stopping criteria of 1,000, 5,000 and 50,000 schedules. The average deviation from the optimal (for J30) and the average deviation from the lower bound (for J60 and J120) were used to measure the performance of the algorithm. All reported results are for 15 runs. The presence of the archive ensures that a solution is only evaluated once even if it is encountered multiple times within the course of evolution. This may cause the algorithm to get stuck in some cases where it is not able to generate enough unique solutions. This problem was only encountered with the J30 instances with limit of 50,000 schedules. An additional stopping criteria of 150,000 (3*50,000) repeated solutions was put in place to ensure that the algorithm terminates if it is no longer able to generate new unique solutions. While this problem was not encountered with J60 and J120, for consistency the same stopping criteria was also put in place for both these problems as well for 50,000 schedule limit.

A. *Problem Complexity Analysis*

The test problems used for this paper are taken from the Project Scheduling Problem Library (PSPLib) [20] which are available online. In this section, we will discuss the factors that contribute to the complexity of problems within PSPLib and how it affects the performance of the proposed algorithm.

1) Size: Problem size refers to the number of activities that need to be scheduled for a given instance. For J30, the optimal values are known, while for the other two, only the lower bounds are available. Optimal values for some instances of J60 and J120 are known, however, for comparison usually the lower bounds are used. As seen from our results, the average deviation increases significantly as the problem size increases.

2) Problem Parameters: The problems within PSPLib are constructed using 3 main parameters. These are network complexity (NC), resource factor (RF) and resource strength (RS). The network complexity reflects the average number of immediate successors of an activity [12]. The resource factor reflects the average percentage of resources required per activity [12]. The resource strength reflects the scarceness of the resource capacities [12].

Based on our observations, the network complexity had little to no effect on the problem difficulty. A high resource factor combined with low resource strength resulted in the most difficult instances for our algorithm to solve. This observation is in correlation with what has been previously discussed in literature with respect to problem complexity. As mentioned by Dorndorf [10], a lower resource strength indicates max-

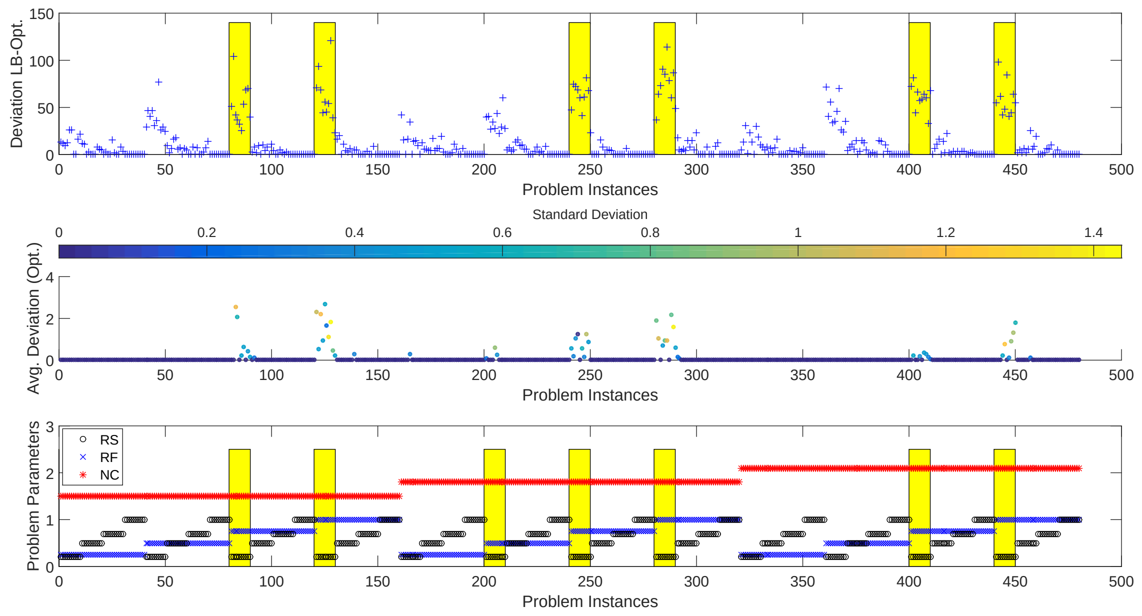


Fig. 3: Performance analysis (5,000 Schedules) of the proposed algorithm for J30 with respect to optimal. RS: Resource Strength, RF: Resource Factor, NC: Network Complexity.

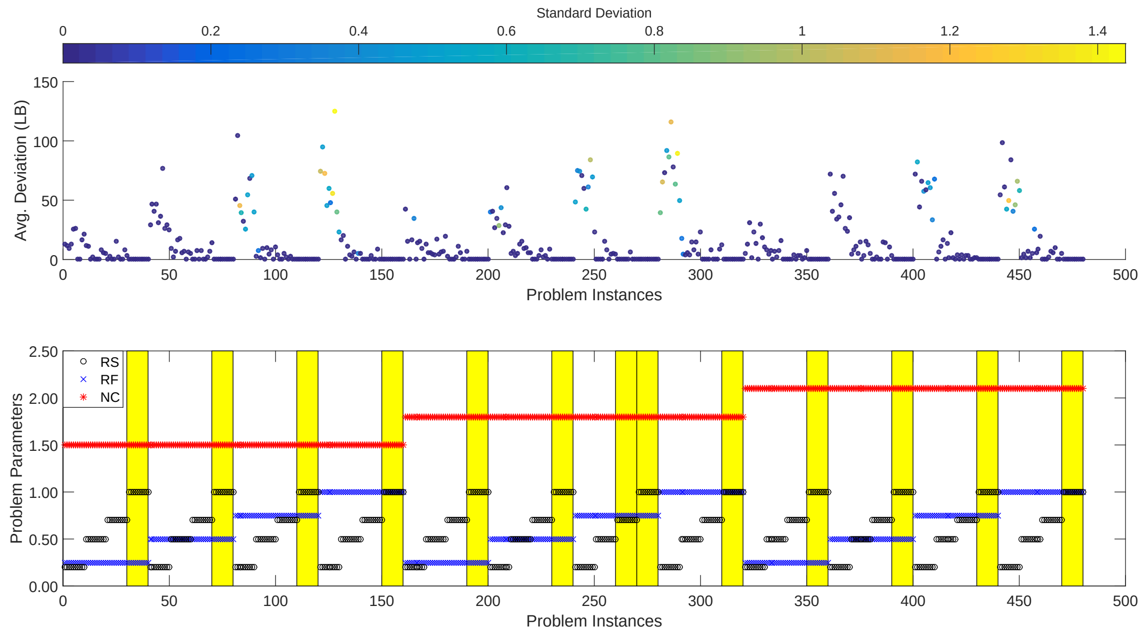


Fig. 4: Performance analysis (5,000 Schedules) of the proposed algorithm for J30 with respect to lower bounds. RS: Resource Strength, RF: Resource Factor, NC: Network Complexity.

imal tightness. This results from minimal feasible resource availability. The complexity is increased when low resource strength is combined with high resource factor. In this situation, a limited availability is combined with high demand for resources. This tightness makes the problem landscape significantly difficult to navigate for an algorithm and introduces the possibility of being stuck at the local optima.

The critical path lower bounds are the project makespans calculated by relaxing all the resource constraints. This is a fairly trivial task. If one were to look at the difference between

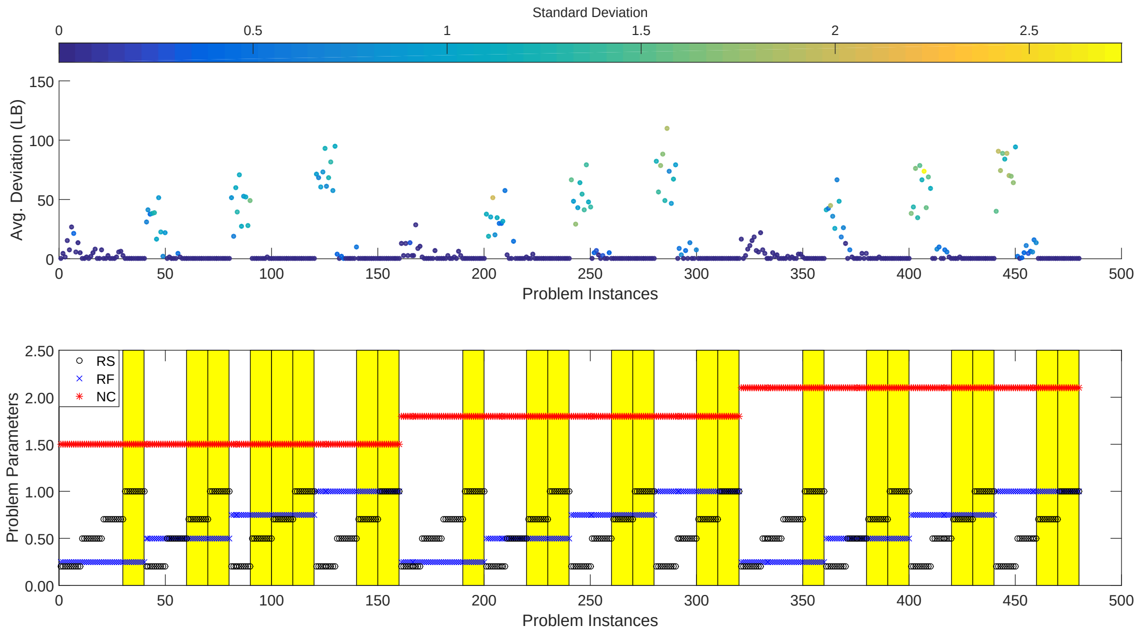


Fig. 5: Performance analysis (5,000 Schedules) of the proposed algorithm for J60 with respect to lower bounds. RS: Resource Strength, RF: Resource Factor, NC: Network Complexity.

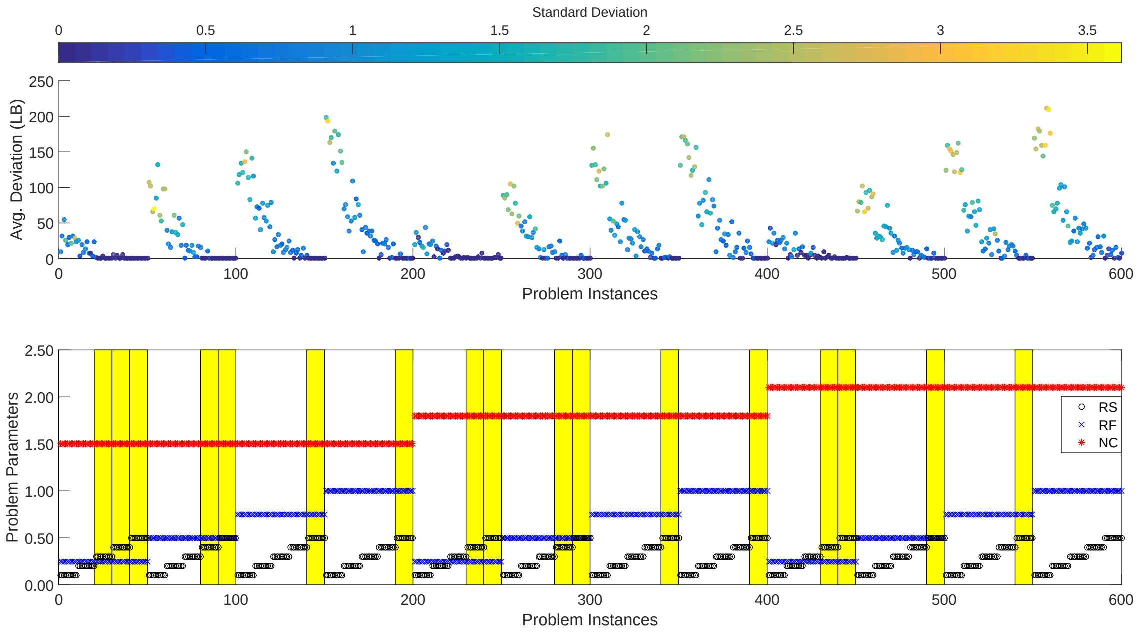


Fig. 6: Performance analysis (5,000 Schedules) of the proposed algorithm for J120 with respect to lower bounds. RS: Resource Strength, RF: Resource Factor, NC: Network Complexity.

the lower bounds and the optimal makespans for J30, the effect of the above mentioned variables would be very clear. The deviation between the optimal and lower bound values are significantly higher for instances where the resource factor is high (0.75, 1.00) and the resource strength is low (0.2). A deviation of 0 from the lower bound makes the problem easy to solve, since it is the same as calculating lower bound. A higher deviation from the lower bound represents a stronger effect of the resource constraints on the problem making it much more difficult to solve.

We have illustrated the above using figures 3, 4, 5, 6. Subplot 1 (top) of Figure 3 shows the percentage deviation between the optimal and lower bound makespans for J30. Since each parameter setting for the PSPLib has 10 representative instances, the highlighted regions (in yellow) indicate sets which have 50% of instances having greater than 50% deviation between optimal and lower bounds. Subplot 2 (middle) shows the performance of our algorithm in terms of average deviation from the optimal for 5,000 schedules with 15 runs. Subplot 3 shows the parameter settings for the J30 problems across all 480 instances. The highlighted regions (in yellow) indicate the parameter settings for which our algorithm faced difficulty in solving. Our algorithm had difficulty in solving problems which had a low resource strength (0.2) and high resource factor (0.75, 1.00). These instances are considered difficult to solve since the effect of the resource constraints are stronger. Subplot 1 is in correlation with the other 2 plots since the deviations between optimal and lower bounds are higher for instances in which resource strength is low and resource factor is high. There is one outlier which can be seen in subplot 3. For this case the resource strength is 0.2 (low) and the resource factor is 0.5 which can be considered relatively high. Our algorithm did not have trouble solving other instances in which this same parameter setting was used which may lead us to believe that this can be due to the effect of the random seed.

Figure 4 presents the same analysis using the lower bounds for J30. Subplot 1 (top) shows the performance of our algorithm in terms of average deviation from the lower bounds for 5,000 schedules. Subplot 2 shows the parameter settings used across the different J30 instances with the highlighted regions indicating parameter settings for sets in which our algorithm was able to achieve deviation of 0 with greater than 80% of the instances. Here we are highlighting instance sets which would be easiest to solve. One can observe that 12 of the 13 highlighted regions have high resource strength (1.00). This indicates high resource availability. We can conclude that for J30, when the resource strength is high, the effect of resource factor is minimal. When availability is high, the level of demand had minimum effect. Most algorithms including ours should be able to solve these instances without much trouble.

Figure 5 illustrates the same concept with J60. Subplot 1 shows the performance of our algorithm in terms of the average deviation from the lower bounds for 5,000 schedules. Subplot 2 shows the parameters settings used across the different problem instances with the highlighted regions (in yellow) indicating sets for which our algorithm was able to achieve deviation of 0 with greater than 80% of the instances. 21 of the 22 highlighted regions have high resource strength (0.70 or 1.00) indicating high availability of resources which implies a weaker resource constraint. This factor makes these instances easy to solve.

For J120, the same analysis is illustrated in Figure 6. Subplot 1 shows the performance of our algorithm in terms of the average deviation from the lower bounds for 5,000 schedules. Subplot 2 shows the parameters settings used across the different problem instances with the highlighted regions (in yellow) indicating sets for which our algorithm was able to achieve deviation of 0 with greater than 50% of the instances. The threshold was reduced to 50% due to the added difficulty incurred as a result of the increase in problem size. These highlighted sets represent problems which would be easiest to solve since the resource constraints are weaker. It must be noted that there may exist other instance sets which can also be classified as "easy", however this will be difficult to determine in the absence of information on the true optimal makespan. Therefore we are limiting our classification of "easy" to those instances where we were able to get deviation of 0.

In the above discussion, we have established the links between problem complexity and the performance of the proposed algorithm. Critical path lower bounds, networks complexity, resource strength and resource factors were used to assess problem complexity. This information can be used to gauge the difficulty associated in solving such problems.

TABLE II: Comparison for J30

Authors	**Schedules**		
	1,000	**5,000**	**50,000**
Mendes et al. [23] (2009)	0.06	0.02	0.01
Kochetov and Stolyar [16] (2003)	0.10	0.04	0.00
Debels and Vanhoucke [8] (2007)	0.12	0.04	0.02
Agarwal et al. [1] (2011)	0.13	0.10	–
Zamani [31] (2013)	0.14	0.04	0.00
Chen et al. [7] (2010)	0.14	0.06	0.01
Zheng and Wang [32] (2015)	0.17	0.06	0.01
This Research	**0.19**	**0.07**	**0.02**
Alcaraz et al. [4] (2004)	0.25	0.06	0.03
Tormos and Lova [26] (2003)	0.25	0.13	0.05
Valls et al. [29] (2008)	0.27	0.06	0.02
Debels et al. [9] (2006)	0.27	0.11	0.01
Chen [6] (2011)	0.29	0.14	0.04
Tormos and Lova [27] (2001)	0.30	0.16	0.07
Alcaraz and Maroto [3] (2001)	0.33	0.12	-
Jia and Seo [15] (2013)	0.34	0.17	–
Valls et al. [28] (2005)	0.34	0.20	0.02
Fang and Wang [11] (2012)	0.36	0.21	0.18
Wang and Fang [30] (2012)	0.38	0.14	–
Hartmann [13] (2002)	0.38	0.22	0.08
Nonobe and Ibaraki [25] (2002)	0.46	0.16	0.05

B. Comparison with Existing Approaches

The results of the proposed algorithm are presented in Table I. The median and mean performance across the 15 runs are almost the same for all cases highlighting stable performance. The standard deviation is consistently low. The results have also consistently improved with the increase in budget.

The performance of the proposed algorithm was compared with 20 other approaches from literature. Tables II, III and IV provide the comparisons for J30, J60 and J120 instances respectively. We have used the results for our best run for comparison. Methods are ranked according to performance on 1,000 evaluations since some of the methods in literature have not reported results for 50,000 evaluations. For J30, our algorithm is ranked 8^{th} for 1,000 evaluations and in a non-dominated sense it is ranked 7^{th} overall (i.e. completely

TABLE I: Results for the proposed algorithm in terms of Average Percentage Deviation from Optimal (J30) and Lower Bound (J60, J120).

Problem	Schedules	Mean	Std Dev.	Best	Median	Worst	Mean Run-Time
J30	**1,000**	0.24	0.02	0.19	0.23	0.28	0.13s
	5,000	0.09	0.01	0.07	0.09	0.12	0.96s
	50,000	0.03	0.01	0.02	0.03	0.04	381.70s
J60	**1,000**	11.90	0.04	11.83	11.89	11.96	0.33s
	5,000	11.31	0.03	11.25	11.32	11.36	2.06s
	50,000	10.91	0.03	10.87	10.90	10.95	366.47s
J120	**1,000**	35.83	0.05	35.75	35.82	35.94	1.03s
	5,000	34.08	0.03	34.04	34.09	34.14	5.84s
	50,000	32.52	0.06	32.45	32.50	32.65	480.22s

TABLE III: Comparison for J60

Authors	Schedules		
	1,000	**5,000**	**50,000**
Debels and Vanhoucke [8] (2007)	11.31	10.95	10.68
Zamani [31] (2013)	11.33	10.94	10.65
Fang and Wang [11] (2012)	11.44	10.87	10.66
Agarwal et al. [1] (2011)	11.51	11.29	–
Valls et al. [29] (2008)	11.56	11.10	10.73
Zheng and Wang [32] (2015)	11.67	10.84	10.64
Kochetov and Stolyar [16] (2003)	11.71	11.17	10.74
Mendes et al. [23] (2009)	11.72	11.04	10.67
Debels et al. [9] (2006)	11.73	11.10	10.71
Chen et al. [7] (2010)	11.75	10.98	10.67
This Research	**11.83**	**11.25**	**10.87**
Tormos and Lova [26] (2003)	11.88	11.62	11.36
Alcaraz et al. [4] (2004)	11.89	11.19	10.84
Wang and Fang [30] (2012)	11.97	11.43	–
Chen [6] (2011)	12.03	11.43	11.00
Tormos and Lova [27] (2001)	12.18	11.87	11.54
Valls et al. [28] (2005)	12.21	11.27	10.74
Hartmann [13] (2002)	12.21	11.70	11.21
Jia and Seo [15] (2013)	12.35	11.96	–
Alcaraz and Maroto [3] (2001)	12.57	11.86	-
Nonobe and Ibaraki [25] (2002)	12.97	12.18	11.58

TABLE IV: Comparison for J120

Authors	Schedules		
	1,000	**5,000**	**50,000**
Debels and Vanhoucke [8] (2007)	33.55	32.18	30.69
Zheng and Wang [32] (2015)	33.87	32.64	31.02
Zamani [31] (2013)	34.02	32.89	31.30
Valls et al. [29] (2008)	34.07	32.54	31.24
Agarwal et al. [1] (2011)	34.65	34.15	–
Kochetov and Stolyar [16] (2003)	34.74	33.36	32.06
Fang and Wang [11] (2012)	34.83	33.20	31.11
Tormos and Lova [26] (2003)	35.01	34.41	33.71
Valls et al. [28] (2005)	35.18	34.02	32.81
Chen et al. [7] (2010)	35.19	32.48	30.56
Debels et al. [9] (2006)	35.22	33.10	31.57
Wang and Fang [30] (2012)	35.44	33.61	–
Chen [6] (2011)	35.71	33.88	32.89
This Research	**35.75**	**34.04**	**32.45**
Mendes et al. [23] (2009)	35.87	33.03	31.44
Tormos and Lova [27] (2001)	36.49	35.81	35.01
Alcaraz et al. [4] (2004)	36.53	33.91	31.49
Jia and Seo [15] (2013)	36.84	35.79	–
Hartmann [13] (2002)	37.19	35.39	33.21
Alcaraz and Maroto [3] (2001)	39.36	36.57	–
Nonobe and Ibaraki [25] (2002)	40.86	37.88	35.85

dominated by 6 other methods). For J60, it is ranked 11^{th} for 1,000 evaluations and in a non-dominated sense it is ranked 10^{th} overall (i.e. completely dominated by 9 other methods). For J120, it is ranked 14^{th} for 1,000 evaluations and 10^{th} overall in a non-dominated sense. It must be noted that a number of methods [8, 6, 31] included in our comparison have not used consistent parameter settings across problems and schedule limits which gives them unfair advantage. It must also be considered that majority of the algorithms ranked better than ours require more parameters in comparison to our approach.

VI. Conclusion

In this paper, a population based stochastic algorithm was proposed for solving the single mode resource constrained project scheduling problem. The algorithm used an adaptive insertion operator for recombination which in the earlier generations allowed for global exploration and in the latter generations allowed exploitation of promising regions. Forward-backward iteration was also used for local search. The results delivered by the algorithm are competitive with existing approaches. A possible shortcoming with the approach is that the run-time is significantly high for 50,000 evaluations. This is because the algorithm only evaluates unique solutions. While this factor may improve accuracy, it adds to significant runtime for 50,000 evaluations. The link between problem complexity and algorithm performance is established which explains exactly why the performance of the algorithm is good or bad for certain instances and more importantly such information can be used to choose appropriate schemes.

Future research would be targeted towards extending our algorithm for other variants of the RCPSP. In particular, the focus will be on practical and realistic constraints.

References

[1] Anurag Agarwal, Selcuk Colak, and Selcuk Erenguc. A neurogenetic approach for the resource-constrained project scheduling problem. *Computers & Operations Research*, 38(1):44 – 50, 2011. ISSN 0305-0548.

[2] Rina Agarwal, M.K. Tiwari, and S.K. Mukherjee. Artificial immune system based approach for solving resource constraint project scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 34(5-6):584–593, 2007. ISSN 0268-3768.

[3] J. Alcaraz and C. Maroto. A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, 102 (1-4):83–109, 2001. ISSN 0254-5330.

[4] J. Alcaraz, C. Maroto, and R. Ruiz. Improving the performance of genetic algorithms for the rcps problem. In *in: Proceedings of the Ninth International Workshop on Project Management and Scheduling*, pages 40–43, 2004.

[5] S. Chand, H. K. Singh, and T. Ray. Finding robust solutions for resource constrained project scheduling problems involving uncertainties. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 225–232, July 2016. doi: 10.1109/CEC.2016.7743799.

[6] Ruey-Maw Chen. Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem. *Expert Systems with Applications*, 38(6):7102 – 7111, 2011. ISSN 0957-4174.

[7] Wang Chen, Yan jun Shi, Hong fei Teng, Xiao ping Lan, and Li chen Hu. An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, 180(6):1031 – 1039, 2010. ISSN 0020-0255.

[8] Dieter Debels and Mario Vanhoucke. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457–469, 2007.

[9] Dieter Debels, Bert De Reyck, Roel Leus, and Mario Vanhoucke. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, 169(2):638 – 653, 2006. ISSN 0377-2217.

[10] Ulrich Dorndorf. *Project Scheduling with Time Windows*. Physica-Verlag HD, 2002.

[11] Chen Fang and Ling Wang. An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem. *Computers & Operations Research*, 39(5):890 – 901, 2012. ISSN 0305-0548.

[12] Snke Hartmann. *Project Scheduling under Limited Resources*. Springer Berlin Heidelberg, 1999.

[13] Snke Hartmann. A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics (NRL)*, 49(5):433–448, 2002. ISSN 1520-6750.

[14] Snke Hartmann and Rainer Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394 – 407, 2000. ISSN 0377-2217.

[15] Qiong Jia and Yoonho Seo. Solving resource-constrained project scheduling problems: Conceptual validation of {FLP} formulation and efficient permutation-based {ABC} computation. *Computers & Operations Research*, 40(8):2037 – 2050, 2013. ISSN 0305-0548.

[16] Yu. A. Kochetov and A. A. Stolyar. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In *in: Proceedings of the 3rd International Workshop of Computer Science and Information Technologies*, pages –, 2003.

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

[18] Rainer Kolisch and Snke Hartmann. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In Jan Wglarz, editor, *Project Scheduling*, volume 14 of *International Series in Operations Research & Management Science*, pages 147–178. Springer US, 1999. ISBN 978-1-4613-7529-6.

[19] Rainer Kolisch and Snke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23 – 37, 2006. ISSN 0377-2217.

[20] Rainer Kolisch and Arno Sprecher. {PSPLIB} - a project scheduling problem library: {OR} software - {ORSEP} operations research software exchange program. *European Journal of Operational Research*, 96(1):205 – 216, 1997. ISSN 0377-2217.

[21] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based milp models for resource-constrained project scheduling problems. *Comput. Oper. Res.*, 38(1):3–13, January 2011. ISSN 0305-0548.

[22] K.Y. Li and R.J. Willis. An iterative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, 56(3):370 – 379, 1992. ISSN 0377-2217.

[23] J.J.M. Mendes, J.F. Gonalves, and M.G.C. Resende. A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research*, 36(1):92 – 109, 2009. ISSN 0305-0548.

[24] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *Evolutionary Computation, IEEE Transactions on*, 6(4):333–346, Aug 2002. ISSN 1089-778X.

[25] Koji Nonobe and Toshihide Ibaraki. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In *Essays and Surveys in Metaheuristics*, volume 15 of *Operations Research/Computer Science Interfaces Series*, pages 557–588. Springer US, 2002. ISBN 978-1-4613-5588-5.

[26] P. Tormos and A. Lova. Integrating heuristics for resource constrained project scheduling: One step forward. Technical report, Department of Statistics and Operations Research, Universidad Politecnica de Valencia, 2003.

[27] Pilar Tormos and Antonio Lova. A competitive heuristic solution technique for resource-constrained project scheduling. *Annals of Operations Research*, 102(1-4):65–81, 2001. ISSN 0254-5330.

[28] Vicente Valls, Francisco Ballestn, and Sacramento Quintanilla. Justification and rcpsp: A technique that pays. *European Journal of Operational Research*, 165(2):375 – 386, 2005. ISSN 0377-2217.

[29] Vicente Valls, Francisco Ballestn, and Sacramento Quintanilla. A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 185(2):495 – 508, 2008. ISSN 0377-2217.

[30] Ling Wang and Chen Fang. A hybrid estimation of distribution algorithm for solving the resource-constrained project scheduling problem. *Expert Systems with Applications*, 39(3):2451 – 2460, 2012. ISSN 0957-4174.

[31] Reza Zamani. A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. *European Journal of Operational Research*, 229(2):552 – 559, 2013. ISSN 0377-2217.

[32] Xiao-Long Zheng and Ling Wang. A multi-agent optimization algorithm for resource constrained project scheduling problem. *Expert Systems with Applications*, 42(1516):6039 – 6049, 2015. ISSN 0957-4174.