A new Discrete Cuckoo Search for Resource Constrained Project Scheduling Problem (rcPSP)

Mehdi Hampaeyan Miandoab

Industrial Engineering Department of Amirkabir University, Tehran, Iran

Abstract
Project schedule problems form a class of important optimization problems, but they are very challenging to solve. In this paper, we formulate a discrete cuckoo search algorithm for scheduling resource constrained project. This algorithm is based on the recently developed Cuckoo Search for continuous optimization. This proposed approach can generate feasible solutions efficiently. We first validate the proposed algorithm against j30, j60, j90 benchmark projects from the PSPLIB site and then compare the results with other algorithm in the literature. Furthermore, we also formulate and propose a new visualization method for displaying optimum solutions that show all events that appear during project makespan. Finally, we discuss the results and their implications for further research.

Keyword: Discrete Cuckoo, Constrained, Project Scheduling, optimization

1. Introduction
Resource Constrained Project Scheduling Problem (RCPSP) is a well known problem that is easy to describe but very difficult to solve, and therefore, it has attracted the attention of many researches over the last few decades. In this context, heuristics are the only option when solving realistically-sized projects [3]. On the other hand, recently developed Cuckoo Search algorithm demonstrate a very simple but efficient approach to solve continuous optimization problems and have had comparable results with the best algorithms in literature. This new heuristic algorithm inspired of a kind of cuckoo birds that show strange behavior in breeding. The other interesting thing in this algorithm is using statistics distributions, like Levy distribution, more efficiently.

Research background
There is a lot of heuristic algorithms that trying to solve resource constrained project scheduling problems as better as possible. Some of these algorithms and its performance is shown in tables in further section for comparing our new algorithm. Heuristic methods like genetic based algorithms are efficient in most hard problems like resource constrained project scheduling problems.

Problem Statement
Main endeavor in solving resource constrained project scheduling problems is finding best start times for each activity in order to completing hole project in minimum possible duration time. There is two kind constrains, first each activity has some precedence activity and cannot start until precedence activities finished completely. Second constriction is that the
availability of each resource in each period of activity duration is limit. So activity start time might be postponed some times because of lacking some needed resources or precedence constrains. In our proposal algorithm we first produce acceptable solutions according presidency constrains and then assign resource for activities.

2. Cuckoo Search
Cuckoo search optimization is one of the newest heuristic algorithms that take its name from a kind of interesting bird Cuckoo, that occupy other birds’ nests for breeding. Almost all animals behaviors, like search for feeding seems be stochastic, but they behaviors follow certain patterns that is similar to some probability distributions. Levy distribution is one of these distributions that can simulate some of animal stochastic behaviors. Cuckoo search algorithm inspired from Cuckoo bird breeding method and also use Levy distribution efficiently to search solution space for finding optimum.

In cuckoo search approach, there are n nest and n initial solutions in each nest. each nest can contain a solution and in each iteration the best solutions keep in nests. by \( P_m \) probability in each iteration the worst nest will challenge to be kept or replaced by new nest that is better according to objective function value. So can be consider that, each nest is an area for local search and keep the best of local area in each iteration and replacing the worst nest by new one is similar mutation operator in GA and used for escaping of local optimums.

3. Resource-Constrained Project Scheduling Problem
The resource-constrained project scheduling problem (RCPSP) can be stated as follows: A single project consists of n activities where each activity has to be processed in order to complete the project. The activities are interrelated by two kinds of constraints. First, precedence constraints imply that activity j cannot be started before all its immediate predecessors have been finished. Second, performing the activities requires resources with limited capacities. Altogether there are a set of R resources. While being processed, activity j requires \( r_{jk} \) units of resource \( k \in R \) at every time instant of its non-empty duration \( p_j \). Resource k has a limited capacity of \( R_k \) at any point in time. The parameters \( p_j, r_{jk}, \) and \( R_k \) are assumed to be non-negative and deterministic. The objective of the RCPSP is to find precedence and resource feasible completion times for all activities such that the makespan of the project is minimal [1].
Though RCPSP has been a very popular benchmark for testing new optimization because and RCPSP problems are usually NP-hard and thus no efficient algorithm exists for solving such problems. However, the last 20 years have witnessed some tremendous improvements of both heuristic and exact solution procedures surveys in the literature. In fact, the RCPSP is often considered as one of the most intractable problems in operations research, it has recently become a popular benchmark for validating latest optimization techniques, and virtually all discrete search algorithms should be tested again this benchmark [1].

Mathematically speaking, therecPSP in math formulation can be formulated as:

\[
\text{Min } t_j
\]

subject to
\[
\begin{align*}
1) & \quad t_j - t_i \geq P_i, \quad \forall j \in S_i \\
2) & \quad \sum_{j \in A_f} l_{rj} \leq t^M_r, \quad r \in R \\
3) & \quad t_j \geq 0, \quad j = 1, \ldots, J
\end{align*}
\]

Where \(t_j\) is an integer and finish time of activity \(j\), and \(A_f\) is the set of in processing activities in period \(f\).

\[A_f = \{ j \mid t_j - \pi_j \div 1 \leq f \leq t_j, j = 1, \ldots, J \}\]

Here \(S_i\) corresponds to the set of activities that have been scheduled before, and \(t^M_r\) is the maximum available amount of resource \(r\) in each period (in the units of day). The first constraint imposes limits on the precedents before every activity, while constraint 2 corresponds to the limitation of every resource.

### 3.1. Generating Feasible Solutions

In the standard RCPSP problem, each solution can be defined as an array or a vector of all activities of project.

![Diagram of a typical project selected from Peterson's 110 test projects.](image)

For example, to schedule the projects shown in Figure 1, we can define a solution as a vector and display solution \(X\) as
Here $X$ is a feasible solution because the first set of constraints is satisfied and the precedence of each activity came before itself. So a feasible solution is an array of activities satisfying all precedent constraints.

One of important components of a solution algorithm is to generate feasible solutions in random way efficiently. In this paper, we propose a discrete cuckoo search to generate new solutions and also to find optimal solutions of the rcPSP.

In particular, each position selects an activity from remain activities and check the precedents; if all precedents are indeed selected and satisfy the constraints, we can accept that activity for current position as a feasible candidate. Otherwise, we select one of those remained precedents that not selected previously before this position. For example, we begin to generate a solution $X$ by selecting a start activity to the first location, and then we select a random activity and check its precedents. Suppose activity 10 is selected as a candidate for the second location, and all precedents of activity 10 are not selected until now and we can choose one of them randomly for second place in $X$. Now suppose we select activity 5 as the candidate so we must check its precedents again. In this manner, we can finally reach activity 1 and because of the only precedent are S activity and is located in the first place we can insert activity 1 in second place. Now the candidate for the third place is 4 and after checking its precedents, it is indeed feasible. Now the candidate for the next place is 5 and then activity 10. Up to this location, our current solution $X$ is

$X = \begin{bmatrix}
S & 1 & 4 & 5 \\
\end{bmatrix}$

Location: 1 2 3 4 5 6 7 8 9 10 11 12 13

Similarly, activity 10 is a candidate for the fifth place, but activity 9 is one of precedents of activity 10 and thus cannot be selected. In this way, we reach activity 3 and hence we can choose it for location 5 and after that activity 9 for location 6, and activity 10 for location 7. So, we first select randomly activity 10 for second place but because of precedence constraint we finally insert it in the 7th place. Now we can randomly select another activity from remaining activities (2, 6, 7, 8 and 11) and follow a similar procedure as outline as above. This simple but yet effective method for generating feasible solutions randomly can be summarized as the pseudo code shown in 0.

1-begin (i=1)
2-From unselected activities, select one randomly as current activity J.
3-If (all precedents of activity J was selected before)
   put activity J in current cell (location), i=i+1,
   else
   select one of unselected precedents randomly as current activity J
   end if
4- Until (i<number of activities) Repeat
5- End
3.2. **Fitness of Feasible Solutions in Scheduling**

After generating feasible solutions, we have to evaluate their suitability or fitness so as to test whether they are near to optimum or not with the ultimate aim to find the global optimum. We must compare generated solutions and be able to find the current solution in each generation. In rcPSP the value of the objective function for each solution is the overall completion time of the last activity. If the last activity is virtual, it can be set to have zero duration, then its start time would be the fitness value of the objective function.

In general, it is necessary to schedule all activities and determine start times. It is clear that we are looking for the earliest start time for each activity considering resource constraints. The earliest start time for activity j, without considering resource constraints, is the maximum of finish/completion times of precedents of j. Since we schedule activities according to sequence on feasible solutions, all precedents of activity j must be scheduled before. Let X is a feasible solution for a simple project shown in figure 1, without virtual start(s) and finish (t) activities.

```
1 2 7 3 9 8 4 6 5 10 11
```

Clearly the start time of first activity 1 is the first day of the project. Since activity 1 is not precedent for activity 2, the start time of activity 2 can be the first time of a project not considering resource constraints. Now we must check resource availability, from the first day and go on to find earliest feasible start time for activity 2. From the history of amount of remain resources in each day, we use an $m \times d$ matrix called RR (Remain Resource) where $m$ is the number of resources engaged in the project and $d$ should be set to a big number that is bigger than the overall possible project makespan. Here $RR(i,d)$ corresponds to the remain amount of resource i on day d. After scheduling each activity, resource demands of that activity should be subtracted from the RR matrix by its duration.

Using such dynamic matrix, we can schedule feasible solutions easily and efficiently. So for finding Earliest Start(ES) time for activity j we can use to the following:

1. Find ES, the maximum of finish times of activity j precedents
2. From ES begin checking resource availability by RR matrix
3.3. **Local Search**
There are various kinds of locale search by using the method that explained in section 2. For local search one way is select some cells of solution and rearrange activities of that cells.

Parameters \(i, j\) and \(l\) are random integers, and proposed algorithm be able to rearrange activities between \(i\) and \(j\), obeying precedents constraints.
4. Algorithm steps

we can summarize the search strategy in some main steps:
1) Choice N nests as Initial population
2) Around each nest do a neighborhood search and save the best solution in the nest
3) By probability $P_m$, find a new nest and replace it with worst nest
4) Do a neighborhood search around the best nest by probability $P_b$
5)

5. implementation

For implementation, imagine a project by 27 activity and 3 resources those are available 6 unit in each period.

An optimum solution and start times for each activities.
display how resource 1 consume along the project time by optimum scheduling

display how resource 2 consume along the project time by optimum scheduling

display how resource 3 consume along the project time by optimum scheduling
6. Comparison with the Best Heuristic for the RCPSP
The results of the our algorithm (RC-CS) show that it obtains a very excellent performance when compared against the best heuristics published and reported in literature by Kolish[7].

Table 3.
Average deviations (%) from optimal makespan—ProGen set J = 30

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>SGS</th>
<th>Reference</th>
<th>1,000</th>
<th>% ADLB 5,000</th>
<th>50,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA-TS-path relinking</td>
<td>Both</td>
<td>Kochetov and Stolyar</td>
<td>0.10</td>
<td>0.04</td>
<td>0.00</td>
</tr>
<tr>
<td>Scatter Search, FBI</td>
<td>Serial</td>
<td>Debels et al.</td>
<td>0.27</td>
<td>0.11</td>
<td>0.01</td>
</tr>
<tr>
<td>GA-hybrid, FBI</td>
<td>Serial</td>
<td>Valls et al.</td>
<td>0.27</td>
<td>0.06</td>
<td>0.02</td>
</tr>
<tr>
<td>GA-FBI</td>
<td>Serial</td>
<td>Valls et al.</td>
<td>0.34</td>
<td>0.20</td>
<td>0.02</td>
</tr>
<tr>
<td>GA–forw.–backw., FBI</td>
<td>Both</td>
<td>Alcaraz et al.</td>
<td>0.25</td>
<td>0.06</td>
<td>0.03</td>
</tr>
<tr>
<td>GA–forw.–backw</td>
<td>Serial</td>
<td>Alcaraz and Maroto</td>
<td>0.33</td>
<td>0.12</td>
<td>–</td>
</tr>
<tr>
<td>Sampling–LFT, FBI</td>
<td>Both</td>
<td>Tormos and Lova</td>
<td>0.25</td>
<td>0.13</td>
<td>0.05</td>
</tr>
<tr>
<td>TS–activity list</td>
<td>Serial</td>
<td>Nonobe and Ibaraki</td>
<td>0.46</td>
<td>0.16</td>
<td>0.05</td>
</tr>
<tr>
<td>Sampling–LFT, FBI</td>
<td>Both</td>
<td>Tormos and Lova</td>
<td>0.30</td>
<td>0.16</td>
<td>0.07</td>
</tr>
<tr>
<td>GA–self-adapting</td>
<td>Both</td>
<td>Hartmann</td>
<td>0.38</td>
<td>0.22</td>
<td>0.08</td>
</tr>
<tr>
<td>GA–activity list</td>
<td>Serial</td>
<td>Hartmann</td>
<td>0.54</td>
<td>0.25</td>
<td>0.08</td>
</tr>
<tr>
<td>Sampling–LFT, FBI</td>
<td>Serial</td>
<td>Tormos and Lova</td>
<td>0.30</td>
<td>0.17</td>
<td>0.09</td>
</tr>
<tr>
<td>TS–activity list</td>
<td>Serial</td>
<td>Klein</td>
<td>0.42</td>
<td>0.17</td>
<td>–</td>
</tr>
<tr>
<td>Sampling–random, FBI</td>
<td>Serial</td>
<td>Valls et al.</td>
<td>0.46</td>
<td>0.28</td>
<td>0.11</td>
</tr>
<tr>
<td>SA–activity list</td>
<td>Serial</td>
<td>Bouleimen and Lecocq</td>
<td>0.38</td>
<td>0.23</td>
<td>–</td>
</tr>
<tr>
<td>GA–late join</td>
<td>Serial</td>
<td>Coelho and Tavares</td>
<td>0.74</td>
<td>0.33</td>
<td>0.16</td>
</tr>
<tr>
<td>Sampling–adaptive</td>
<td>Both</td>
<td>Schirmer</td>
<td>0.65</td>
<td>0.44</td>
<td>–</td>
</tr>
<tr>
<td>RC-CS</td>
<td>–</td>
<td>This Paper</td>
<td>0.63</td>
<td>0.32</td>
<td>0.19</td>
</tr>
</tbody>
</table>

for medium benchmarks j60, we run algorithm only by 1,000 maximum scheduling criteria and because of very excellent result we don't continue by 5,000 and 50,000 because the result won't be effect in ranking.
Table 4. Average deviations (%) from optimal makespan—ProGen set J = 60

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>SGS</th>
<th>Reference</th>
<th>1,000</th>
<th>% ADLB 5,000</th>
<th>50,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC-CS</td>
<td>–</td>
<td>This paper</td>
<td>2.43</td>
<td>Less than 2.43</td>
<td>Less than 2.43</td>
</tr>
<tr>
<td>Scatter search–FBI</td>
<td>Serial</td>
<td>Debels et al.</td>
<td>11.73</td>
<td>11.10</td>
<td>10.71</td>
</tr>
<tr>
<td>GA–hybrid, FBI</td>
<td>Serial</td>
<td>Valls et al.</td>
<td>11.56</td>
<td>11.10</td>
<td>10.73</td>
</tr>
<tr>
<td>GA, TS–path relinking</td>
<td>Both</td>
<td>Kochetov and Stolyar</td>
<td>11.71</td>
<td>11.17</td>
<td>10.74</td>
</tr>
<tr>
<td>GA–FBI</td>
<td>Serial</td>
<td>Valls et al.</td>
<td>12.21</td>
<td>11.27</td>
<td>10.74</td>
</tr>
<tr>
<td>GA–forw.–backw., FBI</td>
<td>Both</td>
<td>Alcaraz et al.</td>
<td>11.89</td>
<td>11.19</td>
<td>10.84</td>
</tr>
<tr>
<td>GA–self-adapting</td>
<td>Both</td>
<td>Hartmann</td>
<td>12.21</td>
<td>11.70</td>
<td>11.21</td>
</tr>
<tr>
<td>GA–activity list</td>
<td>Serial</td>
<td>Hartmann</td>
<td>12.68</td>
<td>11.89</td>
<td>11.23</td>
</tr>
<tr>
<td>Sampling–LFT, FBI</td>
<td>Both</td>
<td>Tormos and Lova</td>
<td>11.88</td>
<td>11.62</td>
<td>11.36</td>
</tr>
<tr>
<td>Sampling–LFT, FBI</td>
<td>Both</td>
<td>Tormos and Lova</td>
<td>12.14</td>
<td>11.82</td>
<td>11.47</td>
</tr>
<tr>
<td>GA–forw.–backw.</td>
<td>Serial</td>
<td>Alcaraz and Maroto</td>
<td>12.57</td>
<td>11.86</td>
<td>–</td>
</tr>
<tr>
<td>SA–activity list</td>
<td>Serial</td>
<td>Bouleimen and Lecocq</td>
<td>12.75</td>
<td>11.90</td>
<td>–</td>
</tr>
<tr>
<td>TS–activity list</td>
<td>Serial</td>
<td>Klein</td>
<td>12.77</td>
<td>12.03</td>
<td>–</td>
</tr>
<tr>
<td>TS–activity list</td>
<td>Serial</td>
<td>Nonobe and Ibaraki</td>
<td>12.97</td>
<td>12.18</td>
<td>11.58</td>
</tr>
<tr>
<td>Sampling–random, FBI</td>
<td>Serial</td>
<td>Valls et al.</td>
<td>12.73</td>
<td>12.35</td>
<td>11.94</td>
</tr>
<tr>
<td>Sampling–adaptive</td>
<td>Both</td>
<td>Schirmer</td>
<td>12.94</td>
<td>12.58</td>
<td>–</td>
</tr>
<tr>
<td>GA–late join</td>
<td>Serial</td>
<td>Coelho and Tavares</td>
<td>13.28</td>
<td>12.63</td>
<td>11.94</td>
</tr>
</tbody>
</table>

Conclusion

In this paper we introduce a new heuristic algorithm that can schedule resource constrained projects in a simple way to implement in MATLAB language programming. Despite easy methods this algorithm result in acceptable outputs that compete with best results known in this field. Using efficient mathematical distributions is one of the main difference in issued algorithm in this paper with other ones like genetic based algorithms. Although this algorithm is an expand for Cuckoo Search algorithm that is powerful in solving continues optimizing problems.

References

[2]. Kwan Woo Kim, MitsuoGenb, GenjiYamazaki ," Hybrid genetic algorithm with fuzzy logic for resource- constrained project scheduling"


